DEPARTMENT of ELECTRICAL ENGINEERING
SCHOOL of ENGINEERING and SCIENCE
NEW YORK UNIVERSITY
UNIVERSITY HEIGHTS BRONX, NEW YORK 10453

15

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) New York University Dept of Electrical Engineering University Heights Bronx, New York 10453	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
	2b. GROUP
3. REPORT TITLE SIMULATION SEQUENCES FOR DISCRETE-TIME SYSTEMS WITH INTERACTIVE PROCESSES	
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Scientific Interim	
5. AUTHOR(S) (First name, middle initial, last name) Dieter J. H. Knollman	
6. REPORT DATE March 1971	7a. TOTAL NO. OF PAGES 137
	7b. NO. OF REFS 14
8a. CONTRACT OR GRANT NO. AFOSR 70-1854	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. 9769	
c. 61102F	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) AFOSR - TR - 71 - 2732
d. 681304	
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.	
11. SUPPLEMENTARY NOTES TECH, OTHER	12. SPONSORING MILITARY ACTIVITY Air Force Office of Scientific Research (NM) 1400 Wilson Boulevard Arlington, Virginia 22209

13. ABSTRACT

One problem encountered in the use of a digital computer for the purpose of simulation is the ordering of simultaneous events. This ordering is relative to the accuracy of the simulation if interaction between simultaneous events occurs. In this thesis the problem of constructing an ordering of behavior algorithms, called a simulation sequence, is discussed. This problem is modeled by directed graphs. An output dependency relation and an algorithm dependency relation are defined. The maximal connected subset partition of the algorithm relation graph is shown to partition the simulation sequence construction problem. An algorithm splitting technique and a selective search routine are presented. These techniques are combined in a general method for constructing simulation sequences.

DD FORM 1 NOV 65 1473

TECHNICAL REPORT 403-18

SIMULATION SEQUENCES FOR DISCRETE-TIME SYSTEMS WITH INTERACTIVE PROCESSES

Dieter J.H. Knollman

March 1971

Prepared By

NEW YORK UNIVERSITY
SCHOOL OF ENGINEERING AND SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING

University Heights
Bronx, New York 10453

For
Air Force Office of Scientific Research
under
Grant Number [illegible]-AFOSR-70-1854

ABSTRACT

One problem encountered in the use of a digital computer for the purpose of simulation is the ordering of simultaneous events. This ordering is relative to the accuracy of the simulation if interaction between simultaneous events occurs. In this thesis the problem of constructing an ordering of behavior algorithms, called a simulation sequence, is discussed. This problem is modeled by directed graphs. An output dependency relation and an algorithm dependency relation are defined. The maximal connected subset partition of the algorithm relation graph is shown to partition the simulation sequence construction problem. An algorithm splitting technique and a selective search routine are presented. These techniques are combined in a general method for constructing simulation sequences.

ACKNOWLEDGMENT

The author wishes to express his debt of gratitude to Professor Herbert Freeman for his guidance and encouragement in the research presented, and to Dr. Alvy Ray Smith III for his corrections of an initial draft of this thesis.

The research reported here was supported in part by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant [illegible]-AFOSR-70-1854 and by the Bell Telephone Laboratories, Inc. through its doctoral support program.

TABLE OF CONTENTS

LIST OF SIMULATION SEQUENCE PROPERTIES

LIST OF ILLUSTRATIONS

LIST OF SYMBOLS

Symbol		Reference Page
α	- algorithm sequence	16
α_y	- output algorithm sequence	36
α_M	- state variable algorithm sequence	36
λ	- algorithm sequence of length zero	16
Σ	- set of input variables	11
Σ_N	- set of network input variables	24
Σ^R	- set of input variables for process p	14
Σ_P	- union of all process - input variable sets	25
σ	- input variable	14
σ^R	- process p input variable	14
σ_N	- network input variable	24
Φ	- empty set	12
A	- algorithm partition	12
A_y	- output algorithm partition	62
B	- family of maximal - connected algorithm subsets	69
C	- maximal algorithm subset	84
D_A	- done algorithm set	94
D_Y	- done output set	94
E_A	- eligible algorithm set	93
E_M	- maximal algorithm set	94
E_Y	- eligible output set	93

Symbol		Reference Page
F	- output algorithm	15
f	- output variable function	15
G	- state-variable algorithm	15
g	- state-variable function	15
H	- algorithm	11
k	- memory-free function	14
K	- connected set of algorithms	67
L	- forward level partition	46
L^c	- backward level partition	52
ℓ	- number of level sets	47
$\ell(y)$	- level of node y	48
M	- set of state variables	11
m	- state variable	14
N	- netwcrk	24
P	- set of processes	24
p	- process	12
$P(y_o,y_n)$	- path from y_o to y_n	41
P(y)	- predecessor set for output y	86
R	- process relation	11
R_A	- process state-variable dependency relation	15
R_B	- process input state-variable dependency relation	15
R_C	- process state-variable output dependency relation	15

Symbol		Reference Page
R_D	- process input-output dependency relation	15
R_N	- network relation	24
R_Y	- output dependency relation	39
$R_Y^{\dagger}$	- transitive closure of R_Y	39
R_Y^C	- converse of relation R_Y	51
R_F	- algorithm dependency relation	61
R_K	- maximal-connected subset relation	69
S	- simulation state vector	20
S(y)	- successor set of output y	86
s_i	- simulation state of variable i	20
V	- value vector	14
v_i	- value of variable i	14
Y	- set of output variables	11
Y_P	- union of all process - output variable sets	25
Y_N	- set of network output variables	24
y	- output variable	14

I. INTRODUCTION

In system design and analysis, simulation is frequently employed to study systems that do not lend themselves to simple analytical methods. In circumstances where experimentation with the actual system is impractical, a digital computer can be used to simulate the operation of the system. Due to the nature of the digital computer several problems arise. Since a digital computer operates in discrete time steps, the continuous time partitions of a system must be approximated by a discrete time system. Furthermore, since a digital computer performs a single operation at a time, concurrent events in a system cannot be simulated in parallel. The ordering of simultaneous events is the subject of this dissertation.

1.1 Statement of the Problem

The system to be simulated is assumed to be characterized by a collection (network) of subsystems (processes). Associated with each process is a set of algorithms specifying the behavior of the process. Since a Mealy type model (output is a function of input and state variables) is employed for each process, any change in process status may propagate instantaneously throughout the network. Since the processes interact the order of execution of process algorithms in updating the simulation time becomes relevant to the accurate simulation of the system.

A simulation sequence is a process algorithm ordering that guarantees an accurate simulation of the system. The ordering of process algorithms to form simulation sequences is the primary subject of this dissertation.

1.2 Literature Survey

Most simulation languages contain an executive routine that calculates simulated time and schedules appropriate event routines. A literature survey of executive routines of simulation languages has been previously published by this writer [1]. This survey is summarized below.

For the purpose of discussion, simulation languages will be divided into two classes, discrete event languages and block diagram languages. A discrete event language is usually an extension of a high level programming language. For example, SIMSCRIPT is an extension of FORTRAN. In a discrete event language a programmer defines the simulation problem by specifying the state of the simulated system and subroutines for each change of state.

The block diagram languages require no programming. In these languages a collection of basic building blocks is supplied with the language. An engineer defines the simulation problem by listing the blocks and the interconnections between blocks.

The problem of simultaneous events was initially ignored. It was presumed that the order of execution of simultaneous events would result in little or no difference since the simulated time step and hence the resulting state changes are usually small. In a comparison of two discrete event languages, GPSS II and SIMSCRIPT, Murphy [2] reports that the two programs produced identical output for a chosen problem after adjustments had been made in random number generation, number rounding, the order in which events are caused and the treatment of simultaneous events. SIMSCRIPT [3] provides no aid in the handling of simultaneous events. GPSS [4] uses a chain propagation technique, that is, if an event triggers a subsequent event, the subsequent event is selected as the next event. The SOL [5] simulation language employs a priority technique. The SIMULA [6] simulation language has a "quasi-parallel" operation mode. In this mode a collection of programs conceptually operates in parallel. In summary, the discrete-event languages place the burden of the accurate simulation of simultaneous events on the programmer.

In the block diagram languages the selection of simultaneous events is made by the simulation program. It will select the order of execution of behavior algorithms for the system blocks. The DAS [7] simulation language executes behavior algorithms for system blocks in the order in which block definitions are read into the program. Engineers using DAS discovered that changing the order of block definitions resulted in different solutions for the same problem. This phenomenon resulted in such a low level of confidence in DAS solutions that DAS was used primarily to obtain scaling information for analog computers. MIDAS [8], a successor of DAS, circumvents the above problem by sorting the block definitions into a predetermined order.

The sorting of block definitions results in an ordering of simultaneous events. The MIDAS sort routine rearranges the blocks so that all blocks have only "known" inputs at all times. Inputs are "known" if they are constants, initial conditions, or outputs of previously ordered blocks. In MIDAS each block has a single output. If all outputs cannot be made "known", then an algebraic loop exists in the block diagram. An algebraic loop, also called a zero delay loop, is an instantaneous cyclic dependency of block output variables on block output variables.

An algebraic loop cannot be made "known" since the cyclic dependency requires that in order to make an output "known", it must have been previously "known." MIDAS will accept an algebraic loop only if an implicit element has been specified. In such a case the output corresponding to the implicit element is made "known," and the sorting operation is continued.

The BLODI [9] language uses a similar sorting technique. BLODI, like MIDAS, allows only one output per block. The BLODI sorting program assigns one binary storage cell for each output. The two states of each output are called "full" and "empty." Initially all outputs which represent inputs to delays are "full" and all others are "empty." A block can be compiled whenever all its inputs are "full" and its output is "empty." When a block is compiled,its output is marked "full." Compilation continues until all blocks have been compiled or until no uncompiled block meets the requirements. In the latter case the remark "closed loop with no delays or all delays" is printed and the problem abandoned. Such a block diagram is called a nonadmissible circuit. The authors [9] state "there is no way to interpret a block diagram corresponding to a nonadmissible circuit. To be sure one could connect physical boxes in such a manner and something would happen. The analysis, however, would involve the precise transient behavior of the devices within the pulse width - information which is not available to the compiler."

In summary, both MIDAS and BLODI sort the block definitions in such a manner that each block uses only constant or previously calculated values. The reader may visualize this ordering as follows:

1. Suppose that each output has a small fixed amount of delay associated with it.
2. Suppose that the block diagram is quiescent and that an input change occurs.

Due to the small amount of delay associated with each output, the input change cannot propagate instantaneously throughout the block diagram but will ripple through the block diagram. If the block diagram is acyclic, that is, contains no cycles,then each block will reach a quiescent state. The blocks that have only block diagram inputs will reach a quiescent state after a single delay unit. The other blocks will reach a quiescent state one delay unit later than the latest input. The order in which the blocks reach a quiescent state is essentially the same as the order produced by the sorting routine. It should be noted that if two or more blocks reach the quiescent state at the same time, the order in which these blocks are sorted is immaterial. This block ordering is called the simulation sequence.

If the block diagram contains a cycle,blocks in the cycle will never reach a quiescent state and hence can never be ordered.

Both MIDAS and BLODI assume that each block contains only a single output. As a consequence of this assumption each block will be included exactly once in the simulation sequence.

The problem of constructing a simulation sequence for multiple output blocks has been studied by Parnas [10]. Parnas defines a multiple output block, called an augmented process, as a 5-tuple $(\Sigma, \Omega, \psi, \mathcal{V}, \Lambda)$ where Σ is a set of input variables, Ω is a set of output variables, ψ is a set of state variables, $\mathcal{V}$ is a single algorithm composed of a set of event description algorithms, and Λ is the instantaneous dependency relation of process output values on process input values. An interconnection of multiple output blocks, called a network, is a quadruple (Δ, Z, Γ, R) where Δ is a set of processes, Z is a set of network inputs, Γ is a set of interconnector variables, and R a set of statements about the network. R relates process outputs with interconnector variables, interconnector variables with process inputs, and network inputs with process inputs.

Parnas considers two types of algorithm applications called process references. The first denoted by $<\Delta>$ is the execution of the process algorithm. The second denoted by $\sim<\Delta>$ is the execution of the process algorithm followed by the restoring of the values of ψ to the values they held before execution. An application of an algorithm will calculate a correct value for an output variable only if all input variables on which this output depends have correct values. A correct value is obtained for the state variables only if all input variables have correct values.

The simulation state of a network is a Boolean vector with one entry corresponding to each process in Δ and one entry corresponding to each member in Γ. All entries are initially "false." An entry corresponding to an element of Γ is "true" if the interconnector variable has a correct value. An entry corresponding to an element of Δ is "true" if all process state variables have correct values. A simulation sequence is a string of process references. The problem of constructing a simulation sequence for multiple output blocks is that of construction a simulation sequence with all "true" final simulation-state entries.

Parnas constructs a minimum-cost simulation seqr ice from a simulation graph. The nodes of the simulation graph are the simulation states of the network. Each branch of the simulation graph is labeled with a process reference. A branch depicts the change in the simulation state that results from the application of a process algorithm. The label of any path in this graph from the "all-false" node to the "all-true" node is a correct simulation sequence.

Parnas proves in his Theorem 3 that there exists an "all-true" node in the simulation graph if and only if there exists no cyclic interconnector dependency. In other words, a correct simulation sequence exists if and only if the network contains no zero-delay loop

In his conclusion Parnas states: "It appears quite easy to generate networks in which the graph used in searching for optimal solutions becomes too large to handle."

1.3 Approach to Solution of Problem

The formulation of the problem is essentially similar to that of Parnas [10]. Functional dependencies will be represented by relations. Only the execution of a given set of algorithms will be considered. Equivalent sequences involving the application of other algorithms will not be considered. Interconnector variables will not be employed. The single-algorithm per process assumption is replaced by a set of algorithms per process assumption. A single algorithm in each process-algorithm set is assumed to update the set of process state-variables. This assumption will lead to a normal form for simulation sequences and will simplify the formation of simulation sequences by removing the dependency of simulation sequences on state variables. It is shown in section 8.1 that this assumption produces no loss of generality.

The process input-output dependency relation is composed with the network interconnection relation to form a network output-dependency relation. This output-dependency relation is represented by a directed graph. The graph is acyclic since no zero-delay loops are allowed in the network.

The theory of directed graphs will be applied to obtain a minimum-length output simulation sequence. The strategy employed is partially to order the set of algorithms. An algorithm-dependency relation is formed by condensing the transitive closure of the output-dependency relation according to the algorithm definition. This algorithm-dependency relation need not be acyclic.

If the algorithm-dependency relation contains reflexive elements, that is, if there exists an algorithm with two or more related outputs, this algorithm is split. A new algorithm-dependency relation is formed using the modified algorithm definition. Although this relation is irreflexive it need not be acyclic. An ordering of the modified algorithm set is generally not possible. An ordering of the maximal connected subsets is, however, always possible. Furthermore a concatenation of minimum-length simulation sequences for maximal connected subsets according to their ordering is shown to form a minimum-length simulation sequence.

The formulation of the problem is discussed in Chapter II. In Chapter III the properties of simulation sequences are derived. A normal form is developed and the problem of constructing a simulation sequence is reduced to the problem of constructing a simulation sequence for output variables. It is shown that state-variable dependencies can be ignored. Level partitions of the set of output variables are investigated in Chapter IV. Algorithm ordering is investigated in Chapter V. Algorithm splicting is investigated in Chapter VI. The formation of simulation sequences is discussed in Chapter VII. Extensions to a larger class of problems are discussed in Chapter VIII.

II. SYSTEM MODEL

The system to be simulated is assumed to be characterized as a network of processes.

2.1 Process Definition

Each process must be capable of modelling a subsystem. A basic system model is shown in Figure 1. In this model a system consists of a set of input variables, Σ; a set of output variables, Y; a set of state variables, M; and a set of memory-free functions, H. For the purpose of simulation, this model is deficient in the following respects:

1. The model implies that the value of each output or state variable depends upon the value of every input and state variable. Although this assumption causes no theoretical difficulty, it places a severe restriction on the problem of obtaining an accurate simulation sequence for the system. In general, an output or state-variable value is only dependent upon some input and state-variable values. The knowledge of the functional dependency of process-variable values will allow the formation of an efficient simulation sequence. The functional dependency is specified by the R relation. Process variables i and j are R-related, denoted by $(i,j) \in R$ or iRj, if the value of variable j depends upon the value of variable i.

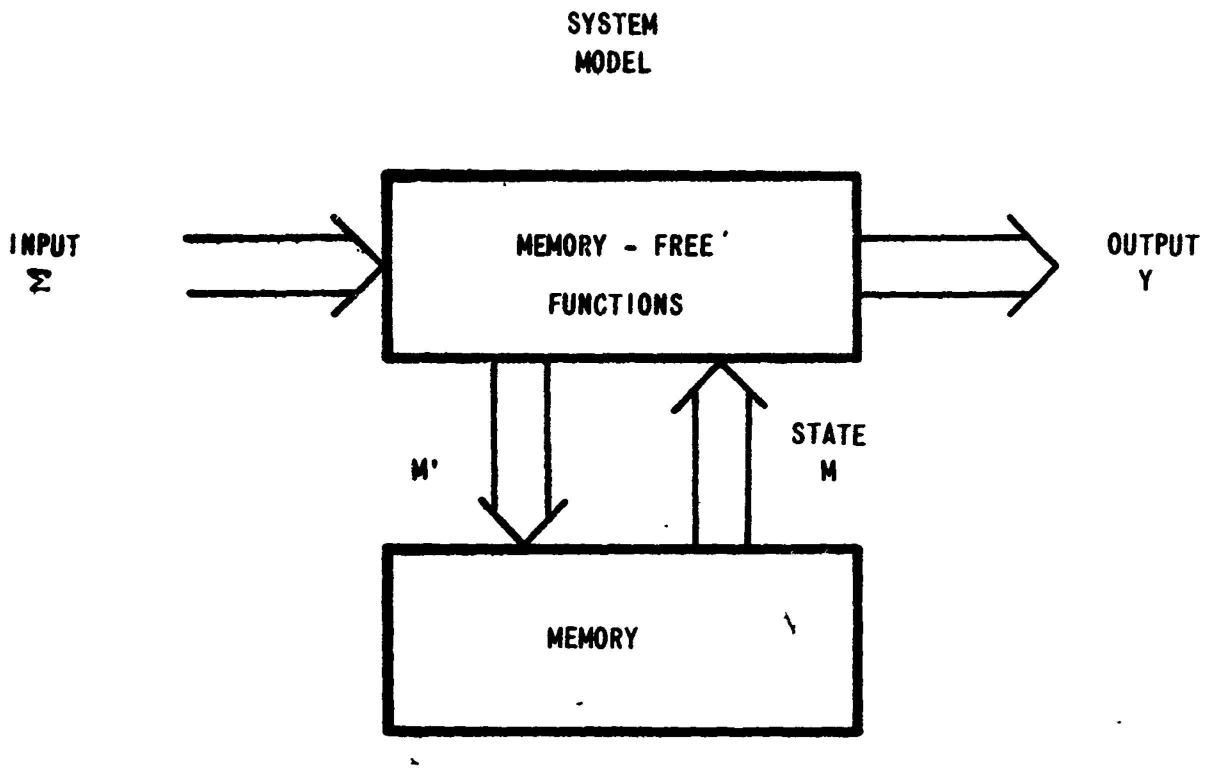


FIGURE 1
SYSTEM MODEL

2. For the purpose of simulation, it is often desirable to combine several functions into a single algorithm. When functions are combined into an algorithm, it is frequently possible to use their similarities to reduce the amount of computation required by eliminating duplicate calculations. The combination of functions into algorithms is specified by the A (algorithm) partition. Initially we will assume that A partitions $Y \cup M$. Later, in order to guarantee that an accurate simulation exists for every process, the A partition will be further restricted.

Formally, a <u>process</u> p is defined to be a quintuple

$p = (\Sigma, Y, M, R, A)$

Σ - finite set of inputs

Y - finite set of outputs

M - finite set of state variables

$R \subseteq (\Sigma \cup M) \times (Y \cup M)$

A - partition of $(Y \cup M)$

The sets of input, output, and state variables are assumed to be disjoint, i.e.,

$$\Sigma \cap Y = \Sigma \cap M = Y \cap M = \Phi \text{ (empty set)}$$

Capital letters will be used to denote sets, and small letter will be used to denote elements of sets. In circumstances, where there exists a need to differentiate between members of a set, subscripts will be used to denote specific elements. Subscripts will be used on sets to denote subsets. Thus $\sigma \in \Sigma$ and $\Sigma_i \subseteq \Sigma$. Superscripts will be used to denote the process to which an element or a set belongs.

Each input, output and state variable has a value as an attribute, i.e.,

v_i - value of variable i

An ordered set of variables, Z, has a value vector, V_Z, as an attribute, i.e.,

$$V_Z = \left(v_{i_1}, v_{i_2}, \ldots, v_{i_n}\right)$$

where

v_{i_j} - value of variable $i_j \in Z$

Corresponding to each output and state variable there exists a function, h, which determines its variable value, i.e.,

$$v_i = h_i\left(V_{\Sigma_i}, V_{M_i}\right) \quad \text{for each } i \in Y \cup M$$

where $\Sigma_i = \{\sigma | \sigma R i\}$

and $M_i = \{m | m R i\}$.

The R relation determines the domain of each function. For the purpose of discussion, it will be convenient to differentiate between output functions and state-variable functions.

f - output function

g - state-variable function

The set of output functions and the set of state variable functions are denoted by F and G.

$$F = \{f\} = \{h_i \mid i \in Y\}$$

$$G = \{g\} = \{h_i \mid i \in M\}$$

By definition there exist one-to-one mappings between F and Y and between G and M. Hence h_i will often be denoted by i. This partition of the set of functions also partitions the domain of the functions and hence specifies a partition of R.

$$R = R_A \cup R_B \cup R_C \cup R_D$$

$$R_A \subseteq M \times M$$

$$R_B \subseteq \Sigma \times M$$

$$R_C \subseteq M \times Y$$

$$R_D \subseteq \Sigma \times Y$$

These process definitions are displayed in Figure 2.

In a simulation problem, separate algorithms may not exist for each function. For various reasons, several functions may have been lumped into a single algorithm. The A partition specifies which functions are to be executed concurrently. Although A is specified as a partition of $(Y \cup M)$, it will be interpreted as a partition of $(F \cup G)$.

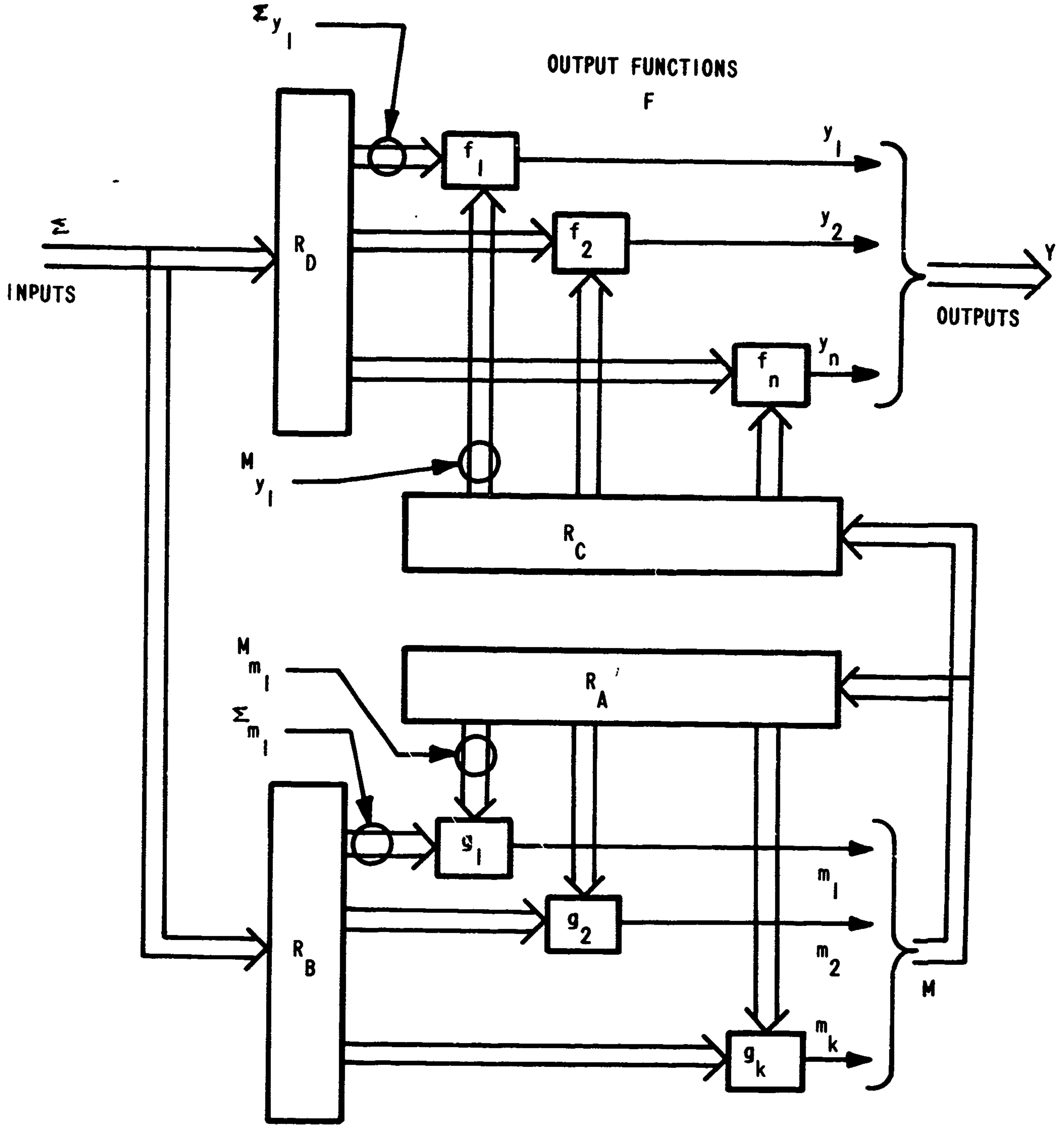
$$A = \{H_i\}$$

$$H_i \cap H_j = \Phi \quad \text{for } i \neq j$$

$$\bigcup_i H_i = F \cup G$$

As a consequence of the A partition, each output and state-variable function is an element of exactly one algorithm. All algorithms are nonempty sets of functions. An empty set of functions will not change any variable values and hence is not of interest.

An algorithm sequence α is an ordered string of algorithms. λ is the algorithm sequence of length zero. Superscripts are used on value vectors to denote the application of an algorithm sequence.



STATE VARIABLE FUNCTIONS
G

FIGURE 2
PROCESS MODEL

V^{λ} - initial value vector

V^{α} - value vector after the application of algorithm sequence α.

In the application of an algorithm all functions in the algorithm are assumed to be applied in parallel, i.e., all functions use the value of variables prior to the application of the algorithm. Thus if the algorithm H is applied after the algorithm sequence α, the resulting values are:

$$v_i^{\alpha H} = \begin{cases} h_i\left(V^{\alpha}_{\Sigma_i}, V^{\alpha}_{M_i}\right) & \text{if} \quad h_i \in H \\ v_i^{\alpha} & \text{otherwise} \end{cases} \qquad \text{where} \quad i \in (Y \cup M)$$

The reader will note that the algorithm H only alters the values of variables corresponding to functions in H. All functions in H use the values of variables prior to the application of H. This differs from a sequential application, in which the initial values are not saved and the values resulting from the functional calculation are employed in subsequent calculations.

A simulation sequence for a process is any algorithm sequence which calculates new values for each output and state variable using only the initial values of the input and state variables. A simulation sequence is formally defined in terms of the simulation state.

The simulation state of a variable is an attribute determined by the variable value. Initially the simulation state of each output and state variable is equal to "0". The simulation state of a variable is equal to "1" if a correct value has been calculated for this variable, and the simulation state is equal to "2" if the variable value is incorrect.

s_i - simulation state of variable $i \in \Sigma \cup Y \cup M$

$$s_i = \begin{cases} 0 & \text{if} \quad v_i = v_i^\lambda \\ 1 & \text{if} \quad v_i = h_i\left(V_{\Sigma_i}^\lambda, V_{M_i}^\lambda\right) \\ 2 & \text{otherwise} \end{cases} \qquad \text{where} \quad i \in Y \cup M$$

The simulation state of a process input variable is determined by the network structure. It is equal to the simulation state of the network variable determining its value.

The simulation state of an ordered set of variables is represented by the simulation-state vector, denoted by a capital S. This vector as the value vector is a function of previously applied algorithms.

S_Z - simulation state vector with a single component associated with each element of Z

S^λ - initial simulation state vector

S^α - simulation state vector after the application of algorithm sequence α.

In the above definition, the simulation state depends upon the variable value and hence upon the variable function. It is conceivable that a correct value may be calculated for a variable although incorrect values are used for the variables on which this variable depends. The above situation is removed by defining the simulation state in terms of a initial simulation state and rules which do not explicitly depend upon variable values.

The application of an algorithm changes the simulation state via the following rule.

$$s_i^{\alpha H} = \begin{cases} s_i^\alpha & \text{if} \quad h_i \notin H \\ 1 & \text{if} \quad (h_i \in H) \text{ and } \left(\sigma R i \Rightarrow s_\sigma = 1\right) \\ & \qquad\qquad\quad \text{ and } \left(m R i \Rightarrow s_m = 0\right) \\ 2 & \text{otherwise} \end{cases}$$

where
$i \in (Y \cup M)$

An algorithm H will not alter the simulation state of variables which are not contained in H. If the variable is included in H, the simulation state is set equal to "1" iff all R-related input variables have "1" simulation state and all R related state variables have "0" simulation state. If the variable is included in H and the above conditions are not satisfied, the simulation state is set equal to "2".

A simulation sequence for a process is any algorithm sequence α such that

$$s_i^\lambda = 0 \quad \text{for each} \quad i \in Y \cup M$$

$$s_i^\lambda = 1 \quad \text{for each} \quad i \in \Sigma$$

and

$$s_i^\alpha = 1 \quad \text{for each} \quad i \in \Sigma \cup Y \cup M$$

A simulation sequence does not exist for all processes. Consider, for example, the process shown in Figure 3. For this process

$$H_1 = \{m_1, y\}$$

and

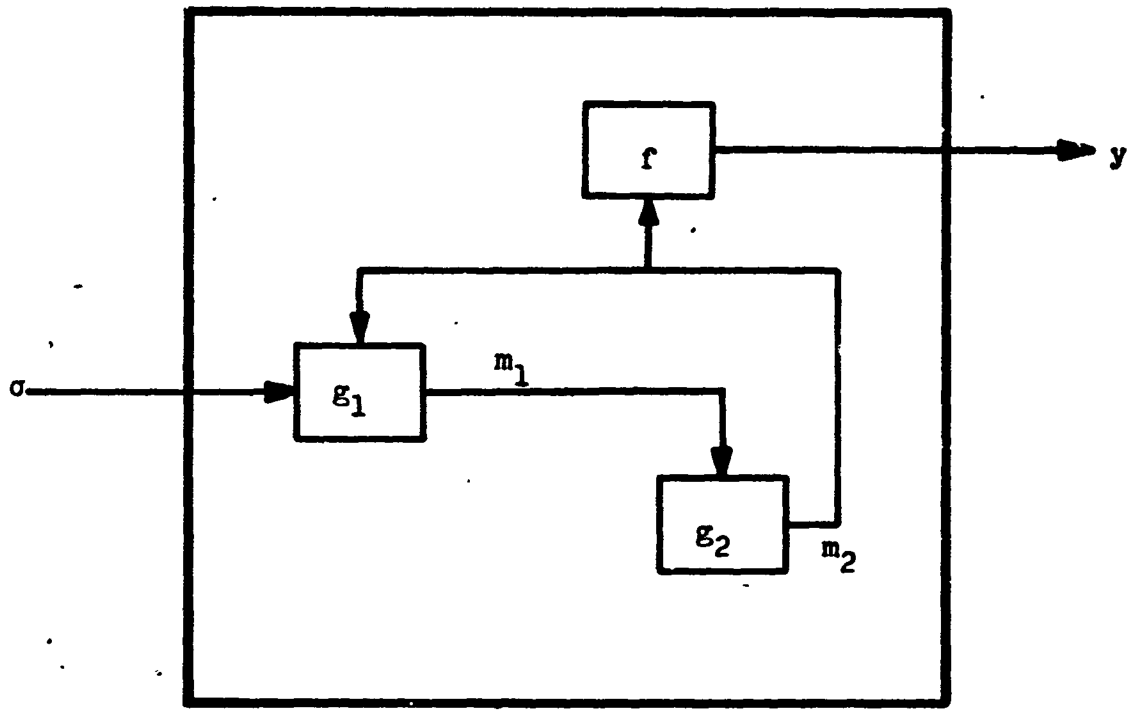
$$H_2 = \{m_2\}.$$

No simulation sequence exists for this process since

$$s_{m_2}^{H_1 H_2} = 2$$

and

$$s_{m_1}^{H_2 H_1} = 2.$$



$p = (\{\sigma\},\{y\}, \{m_1,m_2\}, R, \{H_1,H_2\})$

R: $(\sigma, m_1), (m_1,m_2), (m_2,y), (m_2,m_1)$

$H_1 = \{m_1,y\}$

$H_2 = \{m_2\}$

FIGURE 3

Process For Which No Simulation Sequence Exists

If H_1 is executed first, a new value is computed for m_1. H_2 will use the new value for m_1 and hence will calculate an incorrect value for m_2. If H_2 is executed first, the value of m_1 calculated by H_1 is incorrect. One solution to this dilemma is to store the initial value of all state variables, and only use these stored values in all calculations. This technique requires double storage for all state-variables. An alternative solution is to lump all state variable functions into a single algorithm. In this thesis, it will be assumed that the A partition generates one algorithm equal to G, i.e.,

$$A = \{F_i\} \cup G$$

$$F_i \cap F_j = \Phi$$

$$\bigcup_i F_i = F$$

Under this assumption a simulation sequence exists for every process. The reader will note that

$$\alpha = F_1 F_2 F_3 \ldots F_n G$$

where

$$\bigcup_{k=1}^{n} F_k = F$$

is a simulation sequence for any process.

The simulation state and hence a simulation sequence is independent of variable functions and variable values. Due to the one-to-one correspondence between functions and output and state variables; an algorithm may be represented by a set of output and state variables. This representation of algorithms will be used in the remainder of this thesis. The symbols F, G, and H will still be used to denote algorithms to avoid confusion between algorithms and variable sets.

2.2 Network Definition

A network is a collection of processes. In addition to a list of processes, the network definition must specify the interconnections between processes. This is accomplished by the R_N relation. It relates the entire set of process outputs, denoted by Y_P, to the entire set of process inputs, denoted by Σ_P. The R_N relation also specifies the connection of network inputs Σ_N to process inputs. The network outputs Y_N specify the simulation output. The inclusion of network outputs in the network definition allows the network definition to serve as a process definition. Formally, a network N is defined to be a quadruple

$N = (P,\Sigma_N,Y_N,R_N)$

$P = \{p\}$ - finite set of processes

Σ_N - set of network inputs

$Y_N \subseteq Y_P$ - set of network outputs

$R_N \subseteq (Y_P \cup \Sigma_N) \times \Sigma_P$ - network relation.

A network is thus an interconnection of one or more processes. Processes are assumed to have no variables in common, for example,

$$\Sigma^{p_i} \cap \Sigma^{p_j} = \Phi \text{ (empty set)} \quad \text{for} \quad i \neq j .$$

The set consisting of the union of all process sets of a given type will be denoted by the subscript P.

$$\Sigma_P = \bigcup_P \Sigma^p$$

Network inputs represent external forces on the system. The value of a network input variable is assumed to be given for all time. **A network output is a directly** observable process output.

The network relation R_N specifies the connection of process outputs and network inputs to process inputs.

$(y,\sigma) \in R_N$ or $yR_N\sigma$ if process output y is connected to process input σ.

$(\sigma_N,\sigma) \in R_N$ or $\sigma_N R_N \sigma$ if network input σ_N is connected to process input σ.

A network is <u>complete</u> if every process input is connected to a single process output or network input.

Formally a network is complete if for each $\sigma \in \Sigma_P$ there exists a unique i such that $iR_N\sigma$. The reader will note that the behavior of an incomplete network is not defined. In the following a network is assumed to be complete.

The network relation specifies, for each process input, the process output or network input which determines its value. The simulation state of a process input is equal to the simulation state of the process output or network input to which it is connected, that is, for each

$$(i,\sigma) \in R_N \quad s_\sigma^\alpha = s_i^\alpha .$$

A <u>simulation sequence for a network</u> is any algorithm sequence α such that

$$s_i^\lambda = 0 \quad \text{for each} \quad i \in Y_P \cup M_P$$

$$s_i^\lambda = 1 \quad \text{for each} \quad i \in \Sigma_N$$

and

$$s_i^\alpha = 1 \quad \text{for each} \quad i \in Y_P \cup M_P .$$

2.3 Example Problem

Simulation sequences do not exist for all network Before discussing the properties of a simulation sequence, a network example is presented. A simulation sequence will be constructed for the network of Figure 4.

$N = (\{p_1,p_2,p_3\},\{\sigma_N\},\{y_1,y_7\},R_N)$

R_N: $(\sigma_N,\sigma_2),(y_2,\sigma_5),(y_3,\sigma_9),(y_3,\sigma_6),(y_4,\sigma_1),(y_4,\sigma_4),$
$(y_4,\sigma_7),(y_5,\sigma_8),(y_6,\sigma_3)$

$p_1 = \left(\{\sigma_1,\sigma_2,\sigma_3\},\{y_1,y_2,y_3,\},\{m_1\},R,\left\{\{y_1,y_2,y_3\},\{m_1\}\right\}\right)$

R: $(\sigma_1,y_3),(\sigma_1,m_1),(\sigma_2,m_1),(\sigma_3,y_1),(m_1,y_1),(m_1,y_2)$

$p_2 = \left(\{\sigma_4,\sigma_5,\sigma_6\},\{y_4,y_5\},\{m_2,m_3\},R,\left\{\{y_4,y_5\},\{m_2,m_3\}\right\}\right)$

R: $(\sigma_4,y_5),(\sigma_5,y_5),(\sigma_5,m_2),(\sigma_6,m_2),(m_2,m_3),(m_2,y_4),$
$(m_3,y_4),(m_3,y_5)$

$p_3 = \left(\{\sigma_7,\sigma_8,\sigma_9\},\{y_6,y_7\},\{m_4\},R,\left\{\{y_6,y_7\},\{m_4\}\right\}\right)$

R: $(\sigma_7,y_6),(\sigma_8,y_7),(\sigma_9,y_7),(\sigma_9,m_4),(m_4,m_4),(m_4,y_7)$

The above definitions are displayed in Figure 4. The R_N relation is represented by solid lines and the R relations are represented by broken lines. The R relation is defined to be the union of the process relations. The R

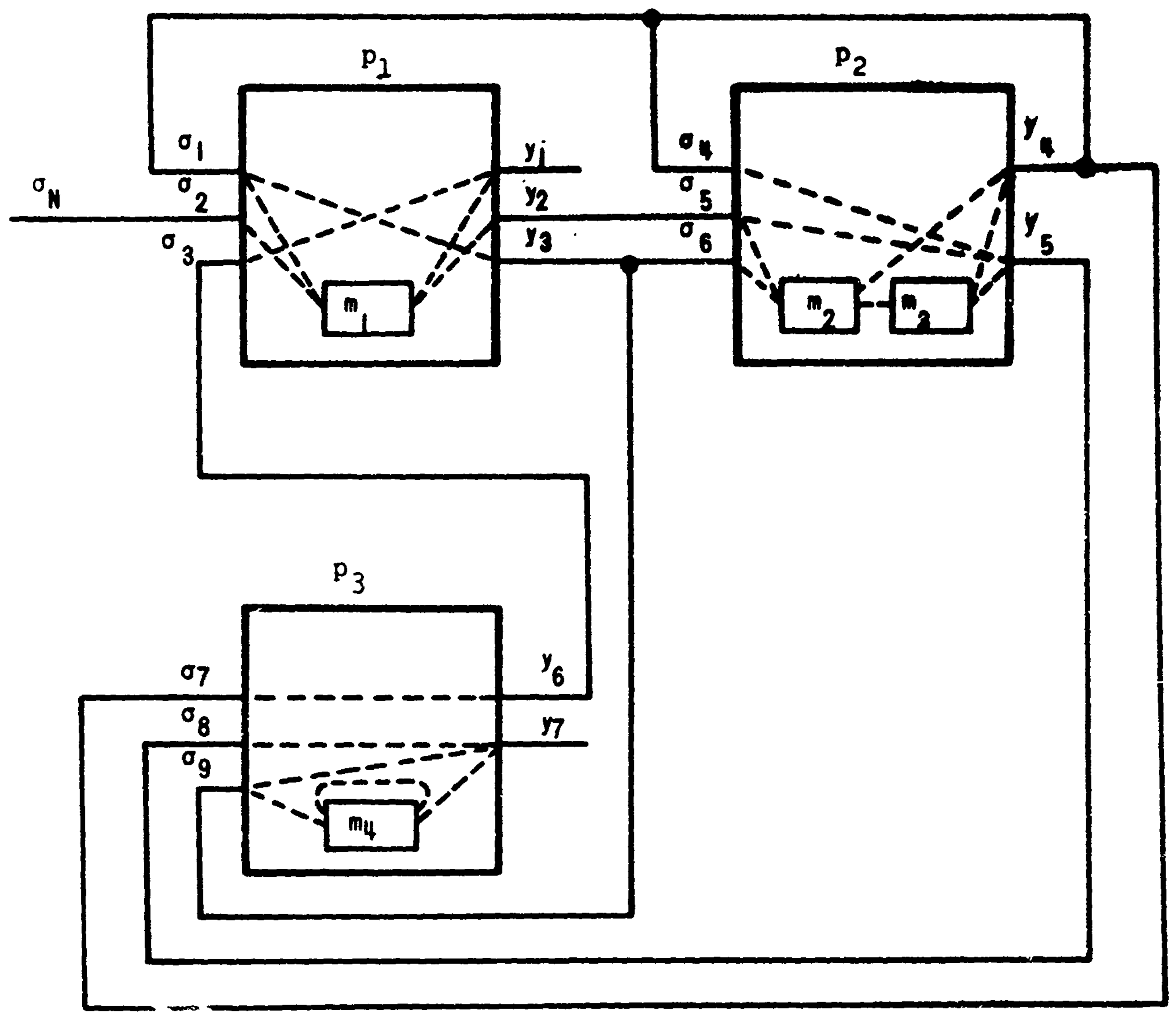


FIGURE 4

ILLUSTRATIVE EXAMPLE NETWORK

relation is thus between ($\Sigma_P \cup M_P$) and ($M_P \cup Y_P$). The <u>composition</u> of R_N and R, written R_NR, is defined as follows

$$w(R_NR)z \iff \text{there exists } \sigma \in \Sigma_P \text{ such that}$$

$$wR_N\sigma \quad \text{and} \quad \sigma Rz$$

The R_NR relation is between ($\Sigma_N \cup Y_P$) and ($M_P \cup Y_P$). The R_NR relation specifies the dependency of process output and state variables on network input and process output variables. For the above example

$$\begin{aligned} R_NR: \;& (\sigma_N,m_1),(y_2,m_2),(y_2,y_5),\\ & (y_3,m_2),(y_3,m_4),(y_3,y_7),(y_5,y_7),\\ & (y_4,y_5),(y_4,m_1),(y_4,y_3),(y_4,y_6),\\ & (y_6,y_1). \end{aligned}$$

The <u>predecessor</u> of an element, z, in the range of a relation R, written R(z), is defined to be

$$R(z) = \{w \mid wRz\}.$$

In the network example

$$R(y_5) = \{\sigma_4,\sigma_5,m_3\}$$

and

$$R_NR(y_5) = \{y_2,y_4\}.$$

The eligible set E is the set of process output and state variables, whose simulation state is not equal to 1, but whose simulation state can be set equal to 1 by the application of a single process algorithm.

$$E = \left\{ v \mid (s_v \neq 1) \quad \text{and} \quad (s_{v'} = 1 \quad \text{for the} \quad v' \in R_N R(v)) \quad \text{and} \quad (s_m = 0 \quad \text{for the} \quad m \in R(v)) \right\}$$

The eligible set is a function of the simulation state. For example, the simulation state vector S is defined as follows:

$$S = \left\{ s_{y_1}, s_{y_2}, \ldots, s_{y_7}, s_{m_1}, \ldots, s_{m_4} \right\}$$

The simulation state of σ_N is always equal to 1 and is not included in S. The intial simulation state vector is

$$S^{\lambda} = (0000000 \quad 0000)$$

and for this simulation state

$$E = \{y_2, y_4, m_3\}.$$

The initial choice of process algorithms is thus reduced to three choices. If G^2 is selected, it will be impossible to calculate correct values for y_4 and y_5 since the initial value of m_2 and m_3 may be altered by the $G^2 = \{m_2, m_3\}$ algorithm. A safe criteria for the selection of process algorithms is to update state variables last. If a minimum-length simulation sequence is desired, F^2 should be selected. The reason for selecting F^2 over F^1 is not obvious and will be developed in succeeding chapters. Selecting $\{y_4,y_5\} = F^2$ as the initial algorithm will calculate a correct value for y_4 and an incorrect value for y_5.

$$S^{F^2} = (0001200 \quad 0000)$$

The application of F^2 removes y_4 from E and adds y_3, y_6 and m_1 to E.

$$E = \{y_2,y_3,y_6,m_1,m_3\}$$

The choice of algorithms is thus between F^1, F^3, G^1 and G^2. The selection of G^1 or G^2 will deny the formation of a simulation sequence. F^1 will correctly update two outputs, and F^3 will correctly update one output. Intuitively, F^1 should thus be preferred over F^3 if a minimum-length simulation sequence is desired. Actually, the number of outputs correctly updated is a poor criteria for the selection of process algorithms. In this case, either selection can lead to a minimum-length simulation sequence. We will select F^3.

$$S^{F^2F^3} = (0001212 \quad 0000)$$

and

$$E = \{y_1,y_2,y_3,m_1,m_3\}.$$

We are forced to select F^1

$$S^{F^2F^3F^1} = (1111212 \quad 0000)$$

and

$$E = \{y_5, m_1, m_2, m_3, m_4\}.$$

All the outputs of p_1 have 1 simulation state entries. G^1 can thus be selected. G^2 and G^3 cannot as yet be selected. Our choices are thus F^2 and G^1. Either choice can lead to a minimum-length sequence. We select G^1.

$$S^{F^2F^3F^1G^1} = (1111212 \quad 1000)$$

and

$$E = \{y_5, m_2, m_3, m_4\}.$$

We are forced to select F^2.

$$S^{F^2F^3F^1G^1F^2} = (1111112 \quad 1000)$$

and

$$E = \{y_7, m_2, m_3, m_4\}.$$

G^3 should still not be selected. F^3 and G^2 are equal choices. We select F^3.

$$S^{F^2F^3F^1G^1F^2F^3} = (1111111 \quad 1000)$$

and

$$E = \{m_2,m_3,m_4\}.$$

G^2 and G^3 are equal choices. We select G^3.

$$S^{F^2F^3F^1G^1F^2F^3G^3} = (1111111 \quad 1001)$$

and

$$E = \{m_2,m_3\}.$$

Only G^2 remains

$$S^{F^2F^3F^1G^1F^2F^3G^3G^2} = (1111111 \quad 1111)$$

$F^2F^3F^1G^1F^2F^3G^3G^2$ is a minimum-length simulation sequence. Other minimum-length simulation sequences are

$$F^2F^3F^1F^2F^3G^2G^3G^1$$

$$F^2F^3F^1G^1F^2G^2F^3G^3$$

$$F^2F^1F^2G^2F^3G^3F^1G^1$$

$$F^2F^1F^2F^3F^1G^1G^2G^3.$$

If F^1 is selected as the initial algorithm, simulation sequences are

$$F^1F^2F^2F^1F^3G^3F^1G^2G^1$$
$$F^1F^2F^3F^1G^1F^3G^3F^2G^2.$$

Although many simulation sequences exist for the example network, there are many networks for which no simulation sequence exists. For example, if (σ_6,y_4) is added to the R relation in the example problem, no simulation sequence will exist for the network. The addition of the (σ_6,y_4) term creates a cyclic chain of input-output and output-input dependencies, namely

$$(\sigma_6,y_4)(y_4,\sigma_1)(\sigma_1,y_3)(y_3,\sigma_6).$$

Due to the cyclic nature of this dependency chain, in order for a variable in the chain to achieve "1" simulation state it must have previously had "1" simulation state. Since the initial simulation state of an output variable is always equal to "0", no output in the chain can achieve a "1" simulation state. Hence no simulation sequence will exist.

III. SIMULATION SEQUENCE PROPERTIES

Although it is possible to develop a large number of simulation-sequence properties, only those properties which provide a clue of the length or structure of a minimum-length simulation sequence are developed in this chapter. In Section 3.1 properties which follow as a consequence of the simulation sequence definition are developed. Properties which depend upon the structure of the network are developed in Section 3.2.

3.1 Basic Properties

For any simulation sequence the initial simulation-state vector always contains all "0" entries. The final simulation state vector contains all "1" entries. A simulation-state entry can only change by the application of an algorithm. Since the set of algorithms partitions the set of network variables, each algorithm must be applied at least once.

Property 1 - Covering

In any simulation sequence each algorithm is included at least once.

Property 2 - Minimum Length

The length of any simulation sequence is greater than or equal to the rank of the algorithm partition.

Property 2 places a lower bound on the length of a simulation sequence. An upper bound on the length of a minimum-length simulation sequence will be developed in Chapter IV.

Property 3 - Normal Form

Given any simulation sequence α, let α_Y denote the algorithm sequence obtained from α by deleting all state-variable algorithms, and let α_M denote an arbitrary ordering of the set of state-variable algorithms. The algorithm sequence $\alpha_Y\alpha_M$ formed by concatenating α_Y and α_M is a simulation sequence.

Proof

In order to set the simulation state of an output equal to "1", the simulation state of every state variable to which this output is R related must be equal to "0", and the simulation state of every output variable on which this output depends must be equal to "1". Delaying the state-variable algorithms will guarantee that the simulation state of every state variable is equal to "0" during the application of the output algorithms. Furthermore since the order of execution of output algorithms has not been altered, every output variable will have "1" simulation state after α_Y.

To set the simulation state of a state variable to "1", all process inputs to which the state variable is R-related must have "1" simulation state and all state variables to which it is R-related must have "0" simulation state. After the output algorithm sequence α_Y has been applied, all outputs have "1" simulation state. Network inputs always have "1" simulation state. Thus all process inputs have "1" simulation state after α_Y. A state variable can only be R related to a state variable in the same process. The assumption of one state variable algorithm per process guarantees that every process state variable will have "0" simulation state prior to the application of the state-variable algorithm corresponding to this process. The simulation state of every output and state variable is thus equal to "1" after $\alpha_Y\alpha_M$. Q.E.D.

As a consequence of Property 3, each state variable algorithm is executed exactly once in α_M and the order of execution of state-variable algorithms within α_M is immaterial. The problem of constructing a simulation sequence is thus reduced to the problem of constructing α_Y for a simulation sequence. The algorithm sequence α_Y will be called an output simulation sequence.

In Section 2.3 a simulation sequence was formed for an example network via a prudent choice of algorithms. The selection of particular algorithm sequences was shown to deny the formation of a simulation sequence. This difficulty does not occur in the formation of output simulation sequences. In fact, if a simulation sequence is known to exist for a network, any output algorithm sequence can be selected as the initial segment of an output simulation sequence.

Property 4 - "1" Simulation State

If the simulation state of an output variable is equal to "1" after some initial segment of a finite-length output algorithm sequence,the application of successive output algorithms will not alter the simulation state.

Proof - By Contradiction

Consider an output y for which $s_y^{\alpha} = 1$ after some output algorithm sequence α. Assume $y \epsilon F$ and that $s_y^{\alpha F} \neq 1$. Clearly $s_y^{\alpha F}$ cannot be equal to "0" and must hence be equal to "2". Thus there exists some $y_1 \epsilon R_N R(y)$ such that $s_{y_1}^{\alpha} \neq 1$. Since s_{y_1} was previously equal to "1," $s_{y_1}^{\alpha}$ must be equal to "2". Hence we may conclude that α is of the following form:

$$\alpha = \alpha_1 F_1 \alpha_2 \quad \text{where} \quad y_1 \epsilon F_1$$

$$s_{y_1}^{\alpha_1} = 1 \quad \text{and} \quad s_{y_1}^{\alpha_1 F_1} = 2.$$

If the length of α_1 equals zero $s_{y_1}^{\alpha_1} = 1$ contradicts the initial condition. If $\#\alpha_1$ is greater than zero, repeat the above argument using y_1 instead of y to obtain α_1'. Since $\#\alpha_1' < \#\alpha_1$, this procedure must terminate and thereby contradict the original assumption. Q.E.D.

Property 4 states that once the simulation state of an output is equal to "1", the application of output algorithms will not alter the simulation state of this output. In the construction of an output simulation sequence it is thus only necessary to achieve a "1" simulation state for each output at some point in the algorithm sequence. Only techniques for attaining a "1" output simulation state need be investigated.

3.2 Structural Properties

In Section 3.1, the basic properties of simulation sequences were presented. These properties are a consequence of the definition of a simulation sequence, and are independent of the structure of the network and process relations. In this section a criterion for the existence of simulation sequences is developed. A simulation sequence exists for a network if α_Y exists. α_M can always be constructed.

In the formation of an output simulation sequence, only the dependency among output variable values need be considered. The output dependency relation R_Y is defined as follows:

$$y_1 R_Y y_2 \quad \text{iff} \quad y_1 R_N R y_2$$

For the example problem of Section 2.3

$$R_Y: \ (y_2,y_5),(y_3,y_7),(y_5,y_7)$$
$$(y_4,y_5),(y_4,y_3),(y_4,y_6),(y_6,y_1)$$

The R_Y relation is the direct output dependency relation. In the example the value of y_7 depends directly upon the values of y_5 and y_3 since $y_5 R_Y y_7$ and $y_3 R_Y y_7$. The value of y_5 depends upon the value of y_2 and the value of y_3 depends upon the value of y_4. The value of y_7 thus depends indirectly upon the values of y_2 and y_4. This indirect dependency relation is the transitive closure of R_Y.

Given any relation R between a set Y and itself, the transitive closure of R, written $R^\dagger$ is defined as follows:

$$R^{\dagger} = \bigcup_{i=1}^{\infty} R^{i} = R \cup (RR) \cup (RRR) \cup \ldots$$

If Y is finite and contains m elements, only the first m terms need be included in the union.

For the example problem

$$R_Y^{\dagger}: (y_2,y_5),(y_2,y_7),(y_3,y_7),$$
$$(y_4,y_5),(y_4,y_3),(y_4,y_6),$$
$$(y_4,y_7),(y_4,y_1),(y_6,y_1)$$

A relation R is called acyclic if $R^{\dagger}$ is irreflexive, i.e., R is acyclic if

$$\text{for each} \quad y \in Y \quad (y,y) \notin R^{\dagger}$$

The relation R is called cyclic if there exists some y such that $yR^{\dagger}y$. The example relation is acyclic.

Any relation R defined on a set Y has a convenient graphical interpretation. The elements of Y are the nodes of the graph, and the elements of R are the edges of the graph.

A <u>directed graph</u> is a system Y = <Y,R>

Y - set of nodes

R - incidence relation

$R \subseteq Y \times Y$

The graph contains an <u>edge</u> from y_1 to y_2 if $(y_1,y_2) \in R$.

The <u>network output graph</u> Y_P is defined as follows:

$$Y_P = \langle Y_P, R_Y \rangle$$

The network output graph for the example problem is shown in Figure 5.

A <u>path</u> from y_0 to y_n in a graph Y, written $P(y_0,y_n)$, is a sequence of edges

$$P(y_0,y_n) = \{(y_0,y_1)(y_1,y_2)\ldots(y_{n-1},y_n)\}$$

such that

$$(y_{i-1},y_i) \in R \quad \text{for all} \quad 0 < i \leq n$$

and

$$\text{except for } y_0 \text{ and } y_n$$

$$i \neq j \text{ implies that } y_i \neq y_j.$$

The length of a path, written #P, is the number of edges in the path. The P(y,y) consisting of the empty sequence of edges is a path of length zero.

A <u>cycle</u> is a path of nonzero length for which the initial node and final node coincide. A graph will contain a cycle if R is cyclic.

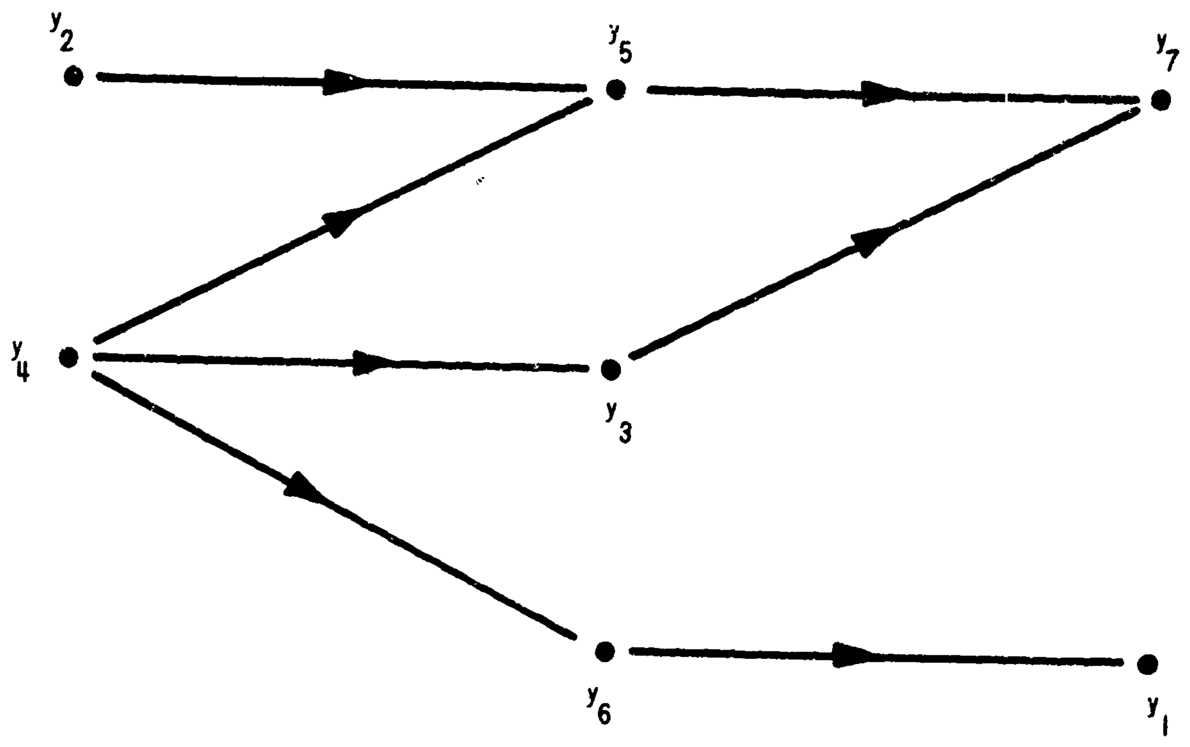


FIGURE 5

NETWORK OUTPUT GRAPH FOR NETWORK
DISPLAYED IN FIGURE 4

The output dependency relation, R_Y, is a precedence relation concerning the "1" simulation state. The reader will note that if $y_1 R_Y y_2$, output y_1 must have a "1" simulation state before the simulation state of y_2 can be set equal to "1". Furthermore if $y_1 \in F_1$ and $y_2 \in F_2$, the output algorithm F_1 must precede the algorithm F_2 in any simulation sequence.

Property 5 - Path Ordering

Given a network output graph $Y_P = <Y_P, R_Y>$ and a path

$$P(y_1,y_n) = \left((y_1,y_2)(y_2,y_3),\ldots,(y_{n-1},y_n)\right)$$

of Y_P, assuming $y_i \in F_i$ for all $1 \leq i \leq n$ then any output simulation is of the form

$$\alpha_0 F_1 \alpha_1 F_2 \alpha_2 F_3 \ldots \alpha_{n-1} F_n \alpha_n$$

Proof

The path from y_1 to y_n implies that for $1 \leq i < n$ the simulation state of y_i must be equal to "1" before the simulation state of y_{i+1} can be set equal to "1", and hence F_i must precede F_{i+1} in any simulation sequence. Q.E.D.

In the example, the network output graph contains four paths of length 2.

$$P(y_2,y_7) = \left((y_2,y_5)(y_5,y_7)\right)$$
$$P(y_4,y_7) = \left((y_4,y_5)(y_5,y_7)\right)$$
$$P(y_4,y_7) = \left((y_4,y_3)(y_3,y_7)\right)$$
$$P(y_4,y_1) = \left((y_4,y_6)(y_6,y_1)\right)$$

By Property 5

$$\alpha = \alpha_0 F^1 \alpha_1 F^2 \alpha_2 F^3 \alpha_3$$

$$\alpha = \alpha_4 F^2 \alpha_5 F^2 \alpha_6 F^3 \alpha_7$$

$$\alpha = \alpha_8 F^2 \alpha_9 F^1 \alpha_{10} F^3 \alpha_{11}$$

$$\alpha = \alpha_{12} F^2 \alpha_{13} F^3 \alpha_{14} F^1 \alpha_{15}$$

Every simulation sequence for the example problem must meet all four of the above restrictions. The sequences

$$\alpha_Y = F^2 F^3 F^1 F^2 F^3$$

and

$$\alpha_Y = F^2 F^1 F^2 F^3 F^1$$

are minimum-length sequences satisfying all four restrictions.

Property 5 is a necessary condition for α_Y of a simulation sequence. Property 5 also provides a sufficient test for an output simulation sequence if an output simulation sequence exists. In other words, if it is known that a simulation sequence exists, any sequence satisfying Property 5 is an output simulation sequence. There are however networks without a simulation sequence, for which an algorithm sequence satisfying Property 5 exists.

Property 6 - Existence

An output simulation sequence α does not exist for a network N if R_Y is cyclic.

Proof - By Contradiction

Assume R_Y is cyclic and that α is an output simulation sequence. R_Y cyclic implies that there exists some y such that $yR_Y^{\dagger}y$. Assume $y \in F$. Since $s_y^{\lambda} = 0$ and $s_y^{\alpha} = 1$, s_y must be set equal to 1 after some F in α. α is thus of the following form:

$$\alpha = \alpha_1 F \alpha_2$$

where

$$s_y^{\alpha_1} \neq 1$$

and

$$s_y^{\alpha_1 F} = 1$$

Since $yR_Y^{\dagger}y$ the sequence $\alpha_1 F$ must contain F at least twice and is of the form

$$\alpha_1 F = \alpha_3 F \alpha_4 F$$

where

$$s_y^{\alpha_3 F} = 1.$$

By Property 4, $s_y^{\alpha_3 F} = 1$ implies that $s_y^{\alpha_1} = 1$. Q.E.D.

Property 6 states that no output simulation sequence exists for a network with a cyclic output dependency relation. In the next chapter an output simulation sequence is shown to exist for any acyclic output dependency relation.

IV. GRAPH LEVELING

In the previous chapter the problem of constructing a simulation sequence was reduced to the problem of forming an output simulation sequence for an acyclic output dependency relation. In this chapter the structure of the network output graph is investigated.

4.1 Level Sets

Since simulation sequences do not exist for cyclic output dependency relations, the network output graph is assumed to be acyclic. Any acyclic directed graph can be graded. In this section two types of gradings will be defined, they are the forward level sets and the backward level sets.

Given any finite acyclic graph Y = <Y,R> the forward level sets L_i are defined as follows:

$$L_0 = \{y \mid R(y) = \Phi\}$$

$$L_i = \left\{ y \mid R(y) \cap L_{i-1} \neq \Phi \text{ and } R(y) \subseteq \bigcup_{k=0}^{i-1} L_k \right\} \quad \text{for } i > 0$$

The forward level sets have a simple graphical interpretation. L_0 contains all nodes which have no predecessors. L_1 contains those nodes which only have edges from nodes in L_0. A node y is in level L_i $(i > 0)$ if there exists an edge from a node in level L_{i-1} to y and every edge entering y comes from a node in a level less than i.

For the example problem, the level sets are

$$L_0 = \{y_2, y_4\}$$

$$L_1 = \{y_3, y_5, y_6\}$$

$$L_2 = \{y_1, y_7\}$$

$$L_i = \Phi \quad \text{for} \quad i \geq 3$$

If Y is finite only a finite number of level sets are nonempty. Thus let ℓ equal the minimum i such that $L_i = \Phi$.

Lemma 4.1 - Only the first ℓ forward level sets are not empty, i.e.,

$$L_i \begin{cases} \neq \Phi & \text{for} \quad i < \ell \\ = \Phi & \text{for} \quad i \geq \ell \end{cases}$$

Proof

By the definition of ℓ, $L_i \neq \Phi$ for $i < \ell$ and $L_i = \Phi$ for $i = \ell$. Assume $L_n = \Phi$ for some $n \geq \ell$. L_{n+1} must also be empty, since any element of L_{n+1} must have a predecessor in L_n. Q.E.D.

As a consequence of lemma 4.1, ℓ is the number of nonempty forward level sets.

Lemma 4.2 - The forward-level sets are disjoint, i.e.,

$$L_i \cap L_j = \Phi \quad \text{for} \quad i \neq j$$

Proof - by contradiction

Assume $y \in L_i \cap L_j$ for some $j > i$. Since $j > 0$, $y \in L_j$ implies that $R(y) \cap L_{j-1} \neq \Phi$. Thus there exists at least one y^1 such that $y^1 \in L_{j-1}$ with $(j-1) \geq i$. But $y \in L_i$ and $y^1 \in R(y)$ implies that $y^1 \in L_k$ for some $k < i$. Q.E.D.

Lemma 4.3

The family of sets L_i for $0 \leq i < \ell$ partitions Y.

Proof

By Lemma 4.2, $L_i \cap L_j = \Phi$ for $i \neq j$. Furthermore, $\bigcup_{i \geq 0} L_i \subseteq Y$ by construction.

It only remains to be proved that $Y \subseteq \bigcup_{i \geq 0} L_i$. Assume $y \in Y$ and $y \notin \bigcup_{i \geq 0} L_i$. Clearly $R(y) \neq \Phi$ and must contain some element $y_1 \notin \bigcup_{i \geq 0} L_i$. Since $y_1 \notin \bigcup_{i \geq 0} L_i$, $R(y_1)$ must contain some element $y_2 \notin \bigcup_{i \geq 0} L_i$ and this procedure may be continued. Since Y is finite, there must exist some $y_n = y_m$ such that

$$y_n R y_{n+1} R \ldots R y_{m-1} R y_m$$

contradicting the acyclic property of R. Q.E.D.

The family of sets L_i thus partitions Y. Each element y of Y is an element of L_i for some i. The level of a node, written $\ell(y)$, is defined as follows:

$$\ell(y) = i \quad \text{if} \quad y \in \Gamma_i$$

Theorem 4.4

In the system $<Y_P, R_Y>$ with acyclic relation R_Y

$$\alpha_Y = L_0 L_1 L_2 \ldots L_{\ell-1}$$

is an output simulation sequence if each Γ_i is a block in the algorithm partition of the given network.

Proof - by induction

In order to prove Theorem 4.4 it is only necessary to show that each output achieves a "1" simulation state at some point in the output algorithm sequence.

Consider any output y such that $\ell(y) = 0$. This output has no predecessors and hence will achieve a "1" simulation state after the algorithm sequence $\alpha = L_0$.

For all $0 \leq n \leq k$ assume that $\ell(y) = n$ implies that output y will achieve a "1" simulation state after the algorithm sequence $\alpha = L_0 L_1 \ldots L_n$. Consider any output y' such that $\ell(y') = k + 1$. Any predecessor y'' of y' is of a level less than the level of y'. Thus $\ell(y'') \leq k$. By inductive hypotheses and Property 4 the simulation state of y'' is equal to "1" after the algorithm sequence $L_0 L_1 \ldots L_k$. The application of L_{k+1} will thus set the simulation state of y' equal to "1".

By Lemma 4.3 each output is included in some forward level set. Each output will thus achieve a "1" simulation state. Q.E.D.

Theorem 4.4 states that the sequence of forward level sets forms an output simulation sequence. This sequence is in fact a minimum-length simulation sequence.

Property 7 - Minimum Length

The length of any output simulation sequence α_Y is greater than or equal to ℓ.

Proof

To prove Property 7, a path $P(y_0, y_{\ell-1})$ of length $\ell - 1$ from a node $y_0 \in L_0$ to a node $y_{\ell-1} \in L_{\ell-1}$ will be constructed. This path includes ℓ output nodes. By Property 5, α_Y must contain at least ℓ algorithms and hence is of length greater than or equal to ℓ.

Consider $y_{\ell-1} \in L_{\ell-1}$, by the definition of the level sets, there exists $y_{\ell-2} \in L_{\ell-2}$ such that $y_{\ell-2} R_Y y_{\ell-1}$. Repeating the above argument, there exists $y_{\ell-3} \in L_{\ell-3}$ such that $y_{\ell-3} R_Y y_{\ell-2}$, etc. Consider the following sequence:

$$y_0 R_Y y_1 R_Y \ldots R_Y y_{\ell-3} R_Y y_{\ell-2} R_Y y_{\ell-1}$$

This sequence is a path containing ℓ nodes. Q.E.D.

As a consequence of Property 1, the sequence of forward level sets forms a minimum-length simulation sequence. It should be noted however that Theorem 4.4 only applies to the special case in which the output algorithm partition and the forward level partition are identical. This is not the case in the example problem.

If an algorithm partition is desired for defining the processes of a network, the forward level partition is an excellent choice. This partition however is not the only partition that will generate a minimum-length output simulation sequence. The following method will generate a partition which is different from the previous partition if two or more minimum-length output simulation sequences exist.

Given a relation $R = \{(y,z)\}$ the converse of R, written R^C, is the relation $R^C = \{(z,y)\}$. If the relation R is represented by a directed graph Y, the converse of R is represented by a directed graph Y^C obtained from Y by reversing the direction of all branches.

Lemma 4.5

R is acyclic iff R^C is acyclic.

Proof

Since $(R^C)^C = R$, it is sufficient to prove that R acyclic implies that R^C is acyclic. Assume R is acyclic and R^C cyclic. R^C cyclic implies there exists a loop

$$y_1 R^C y_2 R^C \ldots R^C y_n R^C y_1$$

The above implies that there exists

$$y_1 R y_n R \ldots R y_2 R y_1$$

contradicting that R is acyclic. Q.E.D.

Given a finite Y and an acyclic relation $R \subseteq Y \times Y$ the <u>backward level sets</u> L_i^C are defined as follows:

$$L_0^C = \{y \mid R^C(y) = \Phi\}$$

$$L_i^C = \left\{ y \mid R^C(y) \cap L_{i-1} \neq \Phi \quad \text{and} \quad R^C(y) \subseteq \bigcup_{k=0}^{i-1} L_k^C \right\}$$

$$\ell^C = \text{minimum } i \quad \text{such that} \quad L_i^C = \Phi$$

The reader will note that the backward level sets also partition Y. The rank of the backward-level set partition, ℓ^C, is equal to ℓ, the rank of the forward-level set partition.

The backward level sets have the following graphical interpretation. A node is in L_0^C if it has no edges leaving it. A node y is in L_i^C if there exists an edge leaving y to a node in L_{i-1}^C and every edge leaving y goes to a node in a level less than i.

For the example problem, the backward level sets are

$$L_0^C = \{y_1,y_7\}$$

$$L_1^C = \{y_3,y_5,y_6\}$$

$$L_2^C = \{y_2,y_4\}$$

In this example, the forward level sets and the backward level sets generate the same partition. This is however not true in general.

Theorem 4.6

In the system $<Y_P,R_Y>$ with acyclic relation R_Y

$$\alpha_Y = L_{\ell-1}^C L_{\ell-2}^C \ldots L_1^C L_0^C$$

is an output simulation sequence if each L_i^C is a block in the algorithm partition of the given network.

Proof

Consider an output $y \in L_{\ell-1}^C$. This output has no predecessors. The simulation state of y is thus equal to "1" after the algorithm sequence $\alpha = L_{\ell-1}^C$.

For all $1 \leq n \leq k$ assume that $y \in L_{\ell-n}$ implies that the simulation state of y is equal to "1" after the algorithm sequence $\alpha = L_{\ell-1}^C L_{\ell-2}^C \ldots L_{\ell-n}^C$.

Consider any output $y \in L^C_{\ell-k}$. Any predecessor of y is in a level greater than $\ell - k$ and will have a "1" simulation state after the algorithm sequence $\alpha = L^C_{\ell-1}L^C_{\ell-2}\cdots L^C_{\ell-k}$. The application of $L^C_{\ell-k-1}$ will thus set the simulation state of y equal to "1". Q.E.D.

Corollary 4.7

The algorithm sequence

$$\alpha_Y = L^C_{\ell-1}L^C_{\ell-2}\cdots L^C_1 L^C_0$$

is a minimum-length output simulation sequence.

The application of the backward level sets to the formation of output simulation sequences is also contingent on the assumption that the algorithm partition and the converse level partition coincide. In the next section output simulation sequences will be constructed from level partitions for arbitrary algorithm partitions.

4.2 Application of Level Partitions

In the previous section, two partitions, which form minimum-length output simulation sequences, were developed. In this section these partitions are applied to generate output simulation sequences for arbitrary algorithm partitions. Before proceeding farther two substitution properties of output simulation sequences are developed.

Property 8 - Substitution

If $\alpha_Y = \alpha_1 F \alpha_2$ is an output simulation sequence and the output algorithm $F \subseteq F_1$ then $\alpha = \alpha_1 F_1 \alpha_2$ is an output simulation sequence.

Proof

By Property 4 the simulation state of an output cannot change from = 1 to ≠ 1. The addition of output functions into a simulation sequence thus cannot change a simulation state from = 1 to ≠ 1. Although this addition may cause some output simulation states to change from ≠ 1 to = 1 at an earlier point in the simulation sequence, all simulation states must be = 1 at the end of the sequence.

Property 9 - Substitution

If $\alpha_Y = \alpha_1 F \alpha_2$ is an output simulation sequence and if output algorithm $F = F_1 \cup F_2$ then $\alpha = \alpha_1 F_1 F_2 \alpha_2$ is an output simulation sequence.

Proof

Consider an output $y \in F \cap F_1$ which has "1" simulation state after the algorithm sequence $\alpha_1 F$. Any predecessor of this output has "1" simulation state after α_1. Output y will thus have "1" simulation state after $\alpha_1 F_1$.

Next consider an output $y \in F \cap F_2$ which has "1" simulation state after the algorithm sequence $\alpha_1 F$. All predecessors of y have "1" simulation state after α_1. By Property 4, all predecessors of y have "1" simulation state after $\alpha_1 F_1$. Output y will thus have "1" simulation state after $\alpha_1 F_1 F_2$. Q.E.D.

In the application of level partitions to the formation of an output simulation sequence, consider the following three uses:

Case 1

A level partition is a refinement of the algorithm partition.

Case 2

The algorithm partition is a refinement of a level partition.

Case 3

General case.

Case 1

A partition $L = \{L_0, L_1, L_2, \ldots, L_{\ell-1}\}$ of a set Y_P is a refinement of a partition $A_Y = \{F_0, F_1, \ldots, F_n\}$ of Y_P if for each $0 \leq i < \ell$ the set L_i is a subset of F_j for some $0 \leq j \leq n$. Each set L_i is thus covered by some F_j.

Theorem 4.8

Given an output algorithm partition A_Y and a forward level partition $L = \{L_0, L_1, \ldots, L_{\ell-1}\}$ if L is a refinement of A_Y such that $L_i \subseteq F_i$ and $F_i \in A_Y$ then the algorithm sequence $\alpha = F_0 F_1 \ldots F_{\ell-1}$ is a minimum-length output simulation sequence.

Proof

By Property 8, α is an output simulation sequence. This sequence is of length ℓ and by Property 7 a minimum-length output simulation sequence. Q.E.D.

A special case of Theorem 4.8 is the case of a single output algorithm. Although this algorithm calculates new values for every output, ℓ applications are required to calculate correct values for each output.

The reader will note that a similar argument can be made for the backward-level partition. In fact, any partition which forms an output simulation sequence can be employed.

Case 2

If the algorithm partition A_Y is a refinement of a forward-level partition L, each output algorithm F_i is a subset of some level set L_j. In this case each level set L_j can be expressed as the union of output algorithms, i.e.,

$$L_j = \bigcup_{k=1}^{m(j)} F_{j,k} \quad \text{where} \quad F_{j,k} \neq F_{j,i} \quad \text{for } k \neq i.$$

Theorem 4.9

Given an output algorithm partition A_Y and a forward level partition $L = \{L_0, L_1, \ldots, L_{\ell-1}\}$ if A_Y is a refinement of L such that for each $0 \leq i < \ell$

$$L_i = \bigcup_{k=1}^{m(i)} F_{i,k},$$

where $F_{i,k} \neq F_{i,j}$ for $k \neq j$, then the sequence

$$\alpha_Y = F_{0,1}F_{0,2}\cdots F_{0,m(0)}F_{1,1}\cdots F_{1,m(1)}\cdots F_{\ell-1,m(\ell-1)}$$

is a minimum-length output simulation sequence.

Proof

By Property 9, the sequence α_Y is an output simulation sequence. In this sequence each algorithm is included exactly once. By Property 2, the sequence α_Y is a minimum-length output simulation sequence. Q.E.D.

A special case of Theorem 4.9 is the case of a separate algorithm for each output. In this case each algorithm is executed exactly once. The order of execution of algorithms corresponding to outputs in the same level set is immaterial. If a multiprocessor computer is available, algorithms corresponding to outputs in the same level can be executed concurrently.

Case 3

Given that R_Y is acyclic, a simulation sequence can be constructed for any algorithm partition as follows:

1. Calculate a level partition
2. Cover each level set with algorithms.

By Properties 8 and 9, the resulting sequence is an output simulation sequence.

Consider the example problem. For this problem the level sets are:

$$L_0 = \{y_2, y_4\}$$

$$L_1 = \{y_3, y_5, y_6\}$$

$$L_2 = \{y_1, y_7\}$$

and the algorithm partition is

$$F^1 = \{y_1, y_2, y_3\}$$

$$F^2 = \{y_4, y_5\}$$

$$F^3 = \{y_6, y_7\}$$

The covers of the level sets are

$$F^1 \cup F^2 \supset L_0$$

$$F^1 \cup F^2 \cup F^3 \supset L_1$$

$$F^1 \cup F^3 \supset L_2$$

Thus

$$\alpha = F^1F^2F^1F^2F^3F^1F^3$$

is an output simulation sequence. The above sequence contains seven algorithms and is not of minimum length since a sequence of length 5 is known to exist for the example problem.

Although the foregoing method does not yield minimum-length simulation sequences, the sequences generated are guaranteed to be simulation sequences and the effort required is minimal.

In this chapter the graphical structure of the network output graph has been examined. The forward and backward level partitions of the set of outputs have been shown to determine output simulation sequences. These partitions form minimum-length simulation sequences for algorithm partitions, which are refinements of or refined by a level partition. The general problem of forming a minimum-length simulation sequence will be discussed in the next chapter.

V. ALGORITHM ORDERING

In the previous chapter two partial orderings were obtained for the set of network outputs. These orderings specify output simulation sequences for any acyclic network and minimum-length output simulation sequences for a special class of algorithm partitions. In this chapter an algorithm dependency relation is obtained from the output dependency relation via a condensation with respect to the algorithm partition.

In Section 5.1 an acyclic algorithm dependency relation is shown to specify a minimum-length output simulation sequence. Unlike the output dependency relation, the algorithm dependency need not be acyclic. A partitioning of the ordering problem is discussed in Section 5.2.

5.1 Algorithm Graph

The R_Y relation specifies the dependency of output values on output values. If $y_1 \in F_1$, $y_2 \in F_2$, and $y_1 R_Y y_2$ the algorithm F_1 must precede the algorithm F_2 in any output simulation sequence. The output dependency relation R_Y in conjunction with the algorithm partition thus specifies a precedence relation among algorithms. This algorithm relation is formally defined as follows:

$$R_F = \{(F_1,F_2) | y_1 R_Y y_2 \text{ for some } y_1 \in F_1 \text{ and some } y_2 \in F_2\}.$$

The relation R_F is the condensation of R_Y with respect to the algorithm partition. This definition of condensation differs from the definition employed by Harary [12] in that reflexive elements are allowed.

The R_F relation is a precedence relation among algorithms. This relation is displayed by the algorithm graph $<A_Y,R_F>$. A node of the algorithm graph is an output algorithm, that is, a nonempty set of outputs. The graph contains a branch from F_1 to F_2 if there exists at least one output $y_2 \in F_2$ whose value directly depends upon some output $y_1 \in F_1$.

For the example problem discussed in the previous chapter

$$A_Y = \{F^1,F^2,F^3\}$$

$$F^1 = \{y_1,y_2,y_3\}$$

$$F^2 = \{y_4,y_5\}$$

and

$$F^3 = \{y_6,y_7\}$$

The network output relation was shown in Figure 5. The algorithm precedence relation R_F is

$$R_F\text{:}\ \ (F^1,F^2),(F^1,F^3),(F^2,F^2),(F^2,F^1),(F^2,F^3),(F^3,F^1)$$

The algorithm graph for the example problem is shown in Figure 6a. The branches of the network output graph determining the branches of the algorithm partition graph are shown in Figure 6b. The algorithm partition graph is formed from the network output graph by merging outputs.

The reader will note that R_F, unlike R_Y, may be cyclic. For the example problem R_Y is acyclic and R_F is cyclic. If R_F is acyclic, a level partition of A_Y can be formed.

Theorem 5.1

If R_F is acyclic, a forward level partition of A_Y determines a minimum-length output simulation sequence.

Proof (by induction)

1. Consider an algorithm $F \in L_0$. This algorithm has no predecessor algorithms. Any output $y \in F$ has no predecessor outputs. The application of F will thus set the simulation state of all outputs in F equal to "1." This argument can be repeated for any additional algorithms in L_0. Thus if α_0 is an arbitrary ordering of the algorithms in L_0, the simulation state of every output in an algorithm contained in α_0 is equal to "1" after α_0.
2. Let α_i denote an arbitrary ordering of the algorithms in L_i. Assume that all outputs in an algorithm contained in the algorithm sequence $\alpha_0\alpha_1\ldots\alpha_{n-1}$

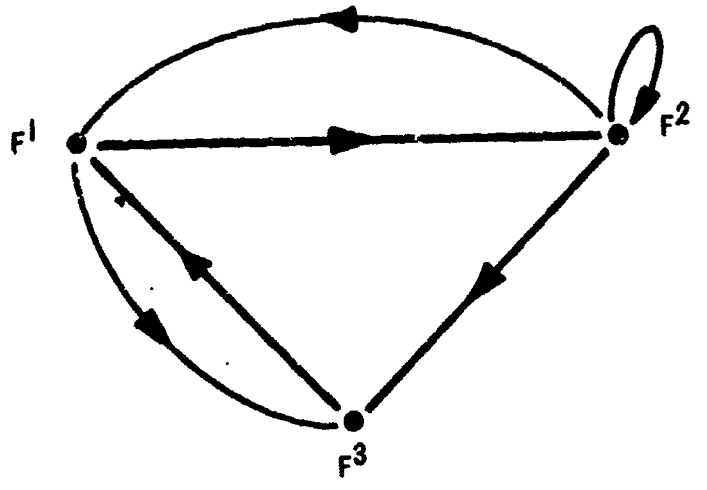


FIGURE 6a

ALGORITHM GRAPH FOR NETWORK DISPLAYED IN FIGURE 4

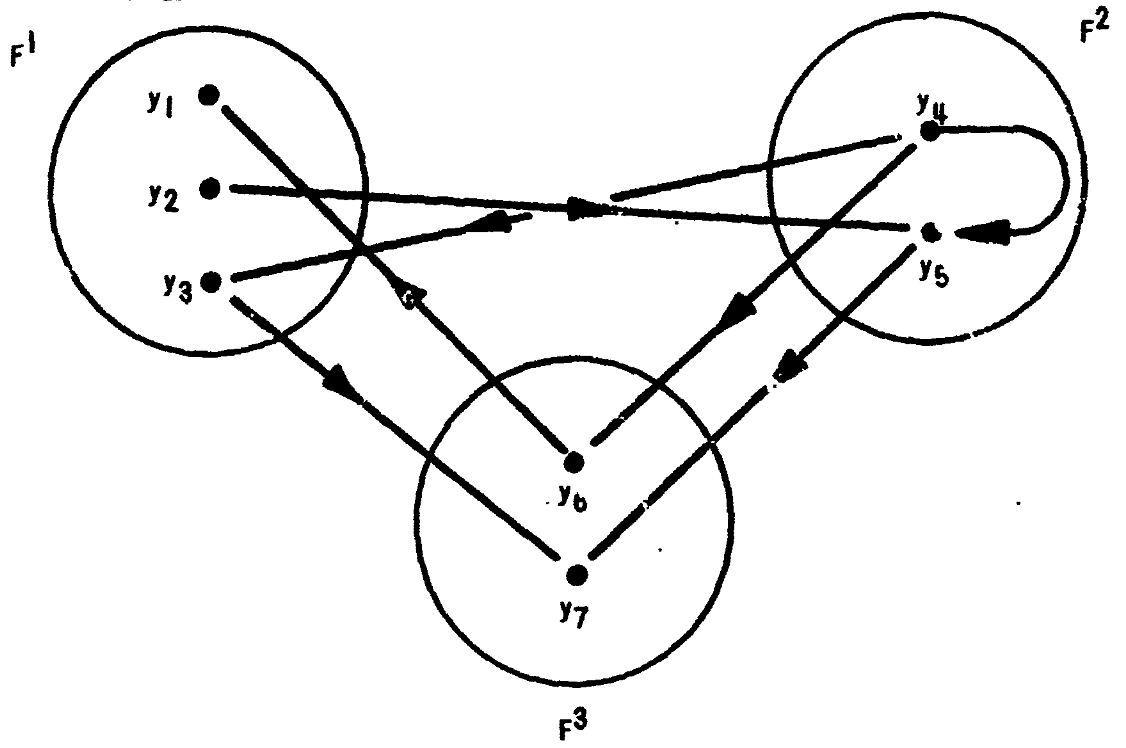


FIGURE 6b

NETWORK OUTPUT GRAPH REDRAWN TO ILLUSTRATE CONDENSATION have "1" simulation state. Consider an output y in an algorithm F in L_n. Output y only depends upon outputs included in algorithms in a level less than n. All of these outputs have "1" simulation state by inductive hypothesis. The application of F will thus set the simulation state of y equal to "1". This argument can be repeated for any additional algorithms in L_n. All outputs in an algorithm in the algorithm sequence $\alpha_0\alpha_1\ldots\alpha_n$ thus will achieve "1" simulation state after the application of this algorithm sequence.

In the level partition each algorithm is in one level and hence occurs once in the algorithm sequence. Since each output is in an algorithm, every output will have "1" simulation state after the algorithm sequence $\alpha_0\alpha_1\ldots\alpha_{\ell-1}$. This sequence is thus an output simulation sequence and by Property 2 a minimum-length output simulation sequence. Q.E.D.

The network output graph displayed in Figure 7 has an acyclic algorithm dependency relation. The sequences $F^1F^2F^4F^3$ and $F^1F^4F^2F^3$ are by Theorem 5.1 minimum-length output simulation sequences.

The reader will note that the backward level sets of A_Y will also form a minimum-length simulation sequence if R_F is acyclic.

If R_F is cyclic, no output simulation sequence of length equal to the rank of the algorithm partitions will

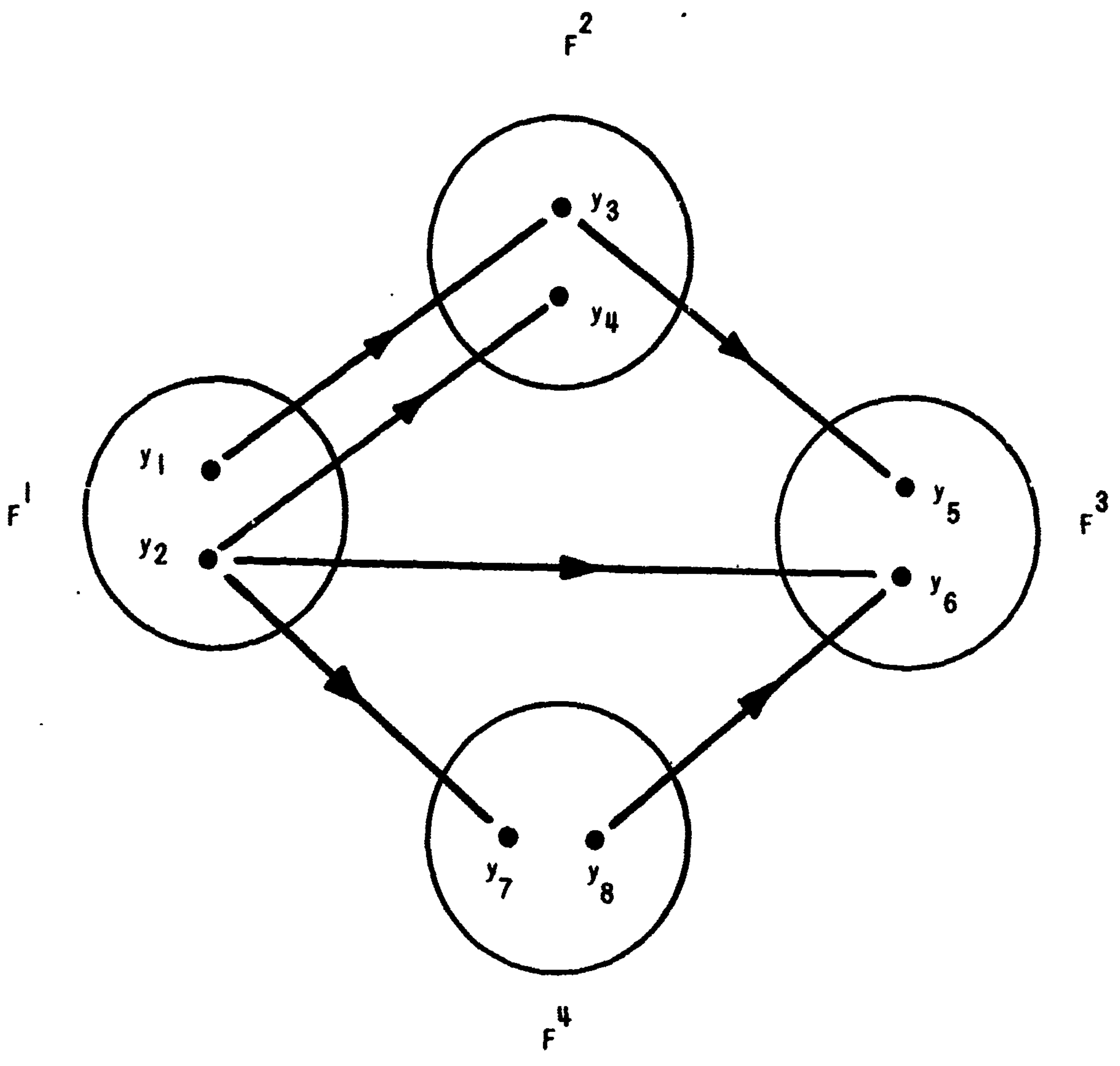


FIGURE 7

NETWORK OUTPUT GRAPH WITH ACYCLIC ALGORITHM DEPENDENCY RELATION exist since at least one algorithm must be applied more than once.

5.2 Algorithm Graph Partitioning

Given any cyclic graph it is always possible to partition this graph into maximal-connected subsets. In this section the maximal-connected subset partition is shown to partition the problem of forming a minimum-length simulation sequence.

A set of algorithms K is connected if for each algorithm F in the set there exist paths in $<A_Y,R_F>$ from F to every other algorithm of the set. An empty set of algorithms is not connected. A single element set is always connected.

In the algorithm graph of Figure 8 the following sets are connected:

$$K_1 = \{F^1,F^2,F^4\}$$

$$K_2 = \{F^1,F^2\}$$

$$K_3 = \{F^5\}$$

$$K_4 = \{F^3,F^6\}$$

$$K_5 = \{F^1\}$$

A maximal connected subset is a connected subset which is not contained in any other connected subset. The

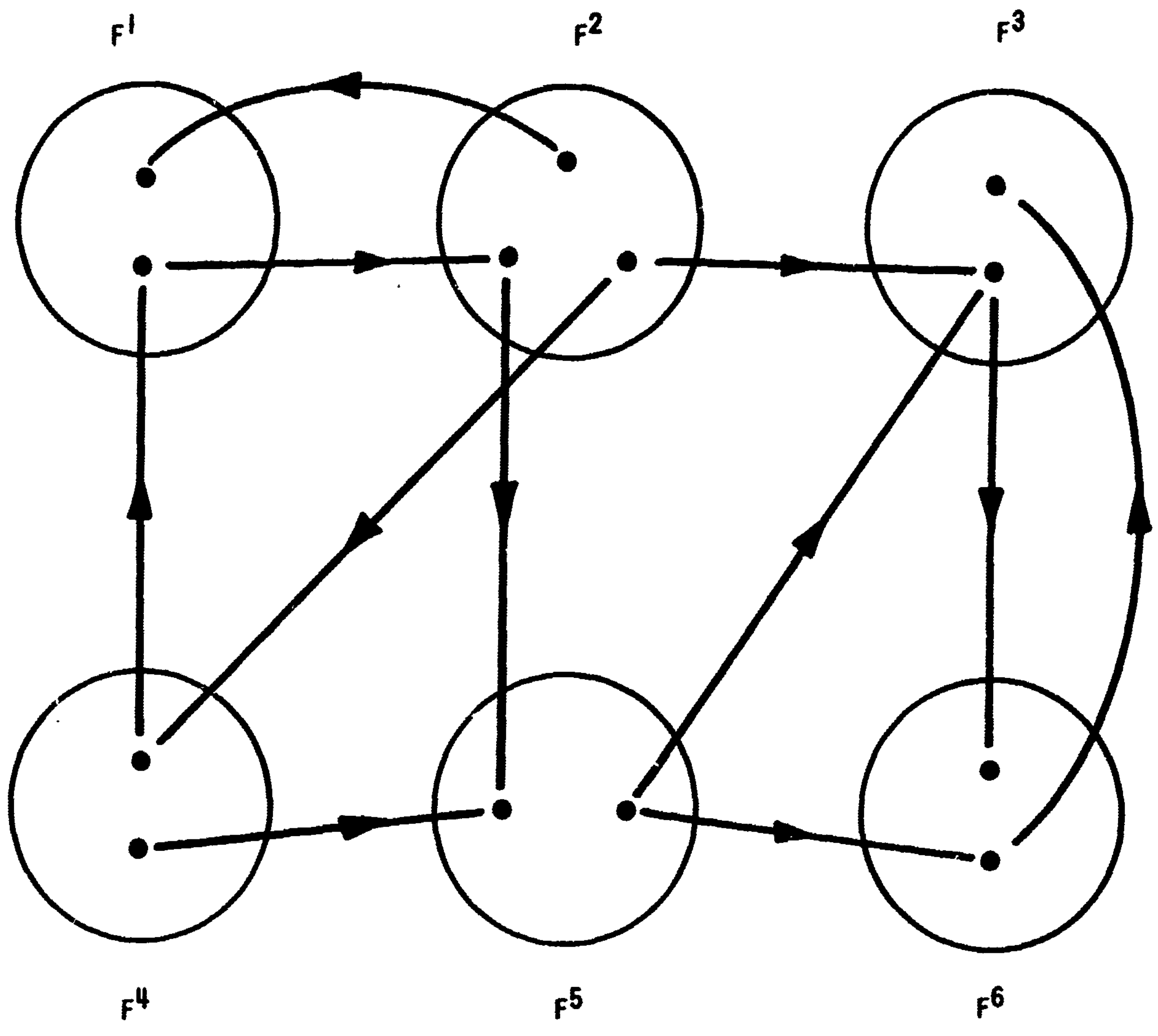


FIGURE 8

NETWORK OUTPUT GRAPH WITH
CYCLIC ALGORITHM DEPENDENCY RELATION

sets K_1, K_3, and K_4 are maximal-connected subsets. The set K_2 is not maximal since it is contained in K_1.

<u>Lemma 5.2</u>

The family of maximal-connected subsets is a partition of the set of algorithms.

<u>Proof</u>

Each algorithm is a connected subset and hence included in a maximal-connected subset. The maximal connected subsets will be shown to be disjoint by contradiction. Assume K_1 and K_2 are maximal-connected subsets with $F_1 \in K_1 \cap K_2$ and $F_2 \in K_1 - K_2$. Claim $K_3 = \{F_2\} \cup K_2$ is connected. Since $F_2 \in K_1$ there exists a path from F_2 to F_1. $F_1 \in K_2$ implies that there exists a path from F_1 to any other algorithm in K_2. Thus there exists a path from F_2 to any algorithm in K_2. Similarly there exists a path from any algorithm in K_2 to F_2. Thus K_3 is connected. Q.E.D.

The R_K relation among maximal connected subsets is defined as follows:

$$K_i R_K K_j \quad \text{iff} \quad i \neq j \quad \text{and} \quad F_i R_F F_j$$

for some $F_i \in K_i$ and some $F_j \in K_j$. R_K is the irreflexive part of the condensation of R_F with respect to the maximal connected subset partition.

For the algorithm graph of Figure 8 the family B of maximal connected subsets is

$$B = \{K_1, K_3, K_4\}$$

and the R_K relation among elements of B is

$$R_K\text{:}\quad (K_1,K_3),\ (K_1,K_4),\ (K_3,K_4)$$

This maximal connected subset relation is displayed in Figure 9.

Lemma 5.3

The relation R_K is acyclic.

Proof

The reader will note that R_K is irreflexive by definition. Assume that R_K is cyclic and that a cycle includes the maximal connected subsets $K_1, K_2, \ldots, K_n$ for $n > 1$. Each K_i is connected. Thus there exists a path between any pair of algorithms in each connected subset. $K_1 R_K K_2$ implies that there exists a path from every algorithm in K_1 to every algorithm in K_2. The R_K cycle thus implies that the union of the connected subsets is a connected subset. Q.E.D.

Since R_K is acyclic, a level ordering can be obtained for the maximal connected subsets.

Theorem 5.4

Given minimum-length simulation sequences for each maximal connected subset of A_Y, a forward level partition of the family of maximal connected subsets determines a minimum-length simulation sequence.

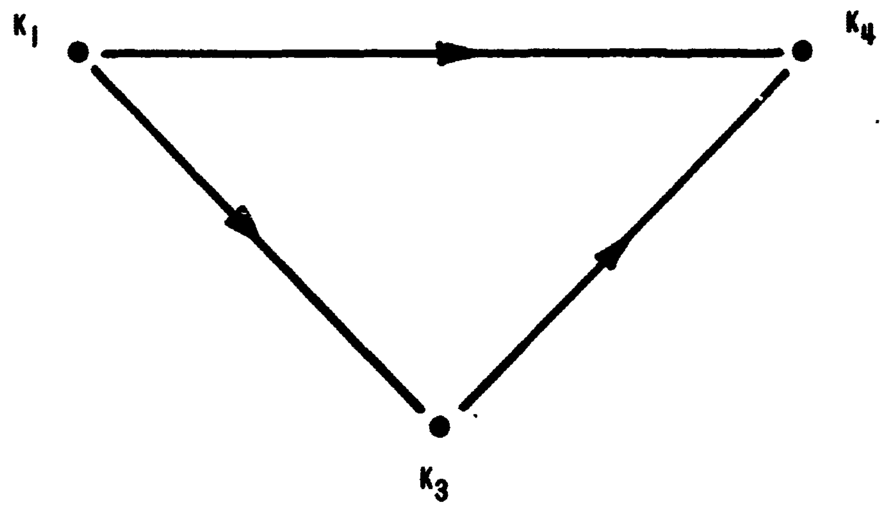


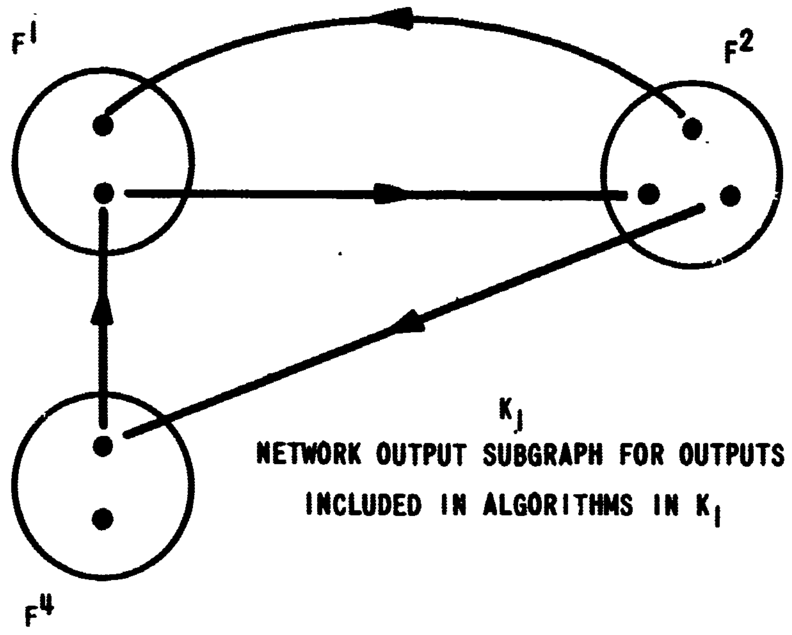
FIGURE 9

MAXIMAL CONNECTED SUBSET RELATION
FOR NETWORK OF FIGURE 8

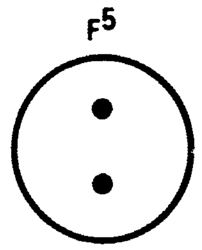
The R_K relation is a precedence relation among maximal connected subsets. The proof of Theorem 5.4 is similar to the proof of Theorem 5.1 and left to the reader. Theorem 5.4 partitions the problem of forming a minimum-length simulation sequence into a collection of smaller problems.

Consider the network output graph of Figure 8. The subgraphs corresponding to each maximal-connected subset are displayed in Figure 10. The reader will note that the sequence $F^2F^4F^1F^2$ is a minimum-length output simulation sequence for K_1, that F^5 is a minimum-length sequence for K_3 and that F^3F^6 is a minimum-length sequence for K_4. As a consequence of Theorem 5.4, $F^2F^4F^1F^2F^5F^3F^6$ is a minimum-length output simulation sequence for the network displayed in Figure 8.

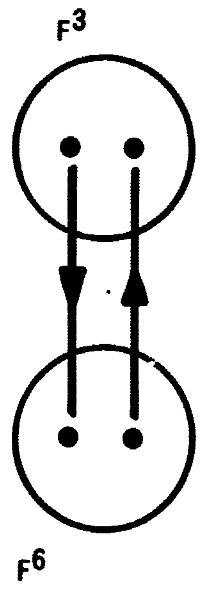
In this chapter an algorithm dependency relation was obtained via a condensation with respect to the algorithm partition. If the algorithm dependency relation is acyclic, a forward level partition of the algorithm graph will determine a minimum-length simulation sequence and if the algorithm graph is cyclic the maximal connected subset partition will split the ordering problem into a collection of smaller problems.



K_1
NETWORK OUTPUT SUBGRAPH FOR OUTPUTS
INCLUDED IN ALGORITHMS IN K_1



K_3
NETWORK OUTPUT SUBGRAPH FOR OUTPUTS
INCLUDED IK ALGORITHMS IN K_3



K_4
NETWORK OUTPUT SUBGRAPH FOR OUTPUTS
INCLUDED IN ALGORITHMS IN K_4

FIGURE 10
NETWORK OUTPUT SUBGRAPHS

VI. ALGORITHM SPLITTING

In the previous chapter an algorithm ordering relation R_F has been defined. If R_F is acyclic a forward level partition of the algorithm graph will determine a minimum-length simulation sequence. If R_F is cyclic at least one algorithm must be included more than once in the output simulation sequence. In this case some outputs of the algorithm will achieve a "1" simulation state in the first application and other outputs in the algorithm will achieve a "1" simulation state by subsequent applications of this algorithm. If an algorithm is applied more than once in a simulation sequence, this algorithm is equivalent to a set of algorithms. One algorithm in this set includes those outputs which achieve a "1" simulation state by the first application of the algorithm, another algorithm includes those outputs which achieve a "1" simulation state by the second application of the algorithm, etc. Thus if it is known that an algorithm must be included n times in a simulation sequence, this algorithm can be split into n subsets. Such a split is developed in this chapter.

In Section 6.1 output simulation sequences will be characterized as refinements of algorithm partitions. In Section 6.2 a criteria for partitioning an algorithm is developed. The algorithm partitions developed in Section 6.2 are not unique and a representation for this family of level partitions is presented in Section 6.3. Refinements of the representation are developed in Section 6.4.

6.1 Output Partitions

In this section output simulation sequences will be characterized as partitions of Y_P. Given any minimum-length output simulation sequence

$$\alpha_Y = F_1 F_2 F_3 \ldots F_n$$

let

$$Y_1 = \left\{ y \middle| s_y^{F_1} = 1 \right\} \quad \text{and for} \quad 1 < i \leq n$$

let

$$Y_i = \left\{ y \middle| s_y^{F_1 F_2 \ldots F_i} = 1 \quad \text{and} \quad s_y^{F_1 F_2 \ldots F_{i-1}} \neq 1 \right\}.$$

Y_i is the set of outputs whose simulation state is set equal to "1" by the application of the ith algorithm. For the example problem of Section 2.3

$$\alpha_Y = F^2 F^3 F^1 F^2 F^3$$

is an output simulation sequence. For this sequence

$$Y_1 = \{y_4\}$$

$$Y_2 = \{y_6\}$$

$$Y_3 = \{y_1, y_2, y_3\}$$

$$Y_4 = \{y_5\}$$

$$Y_5 = \{y_7\}$$

In this section, the family of Y_i sets is shown to be an output algorithm partition refinement which forms an output simulation sequence.

Theorem 6.1

The family of sets Y_i for $1 \leq i \leq n$ is a refinement of the output algorithm partition A_Y.

Proof

By definition, Y_i is the set of outputs whose simulation state is set equal to "1" by the application of F_i. Thus $Y_i \subsetneq F_i$ for each $1 \leq i \leq n$.

Since α_Y is a simulation sequence, the simulation state of each output $y \in Y_P$ must change from not equal to "1" to equal to "1" after the application of some algorithm. Each output is thus in an element of Y_i for some $1 \leq i \leq n$.

Q.E.D.

Theorem 6.2

The sequence

$$\alpha = Y_1 Y_2 \ldots Y_n$$

is an output simulation sequence.

Proof

Since the family Y_i partitions Y_P, each output $y \in Y_P$ is an element of Y_j for some j and will achieve a "1" simulation state after the application of Y_j. By Property 4, this output will have "1" simulation state after the sequence $Y_1 Y_2 \ldots Y_n$. Q.E.D.

Given any minimum-length output simulation sequence of length n, a partition of Y_P of rank n can be extracted from the simulation sequence such that the partition will form a simulation sequence. In this simulation sequence no algorithm is executed more than once. Thus for every network, for which a simulation sequence exists (R_Y is acyclic), there exists a partition which is a refinement of A_Y and which can be ordered to form a minimum-length output simulation sequence. The problem of finding a minimum-length output simulation sequence is thus equivalent to the problem of finding a minimum-rank partition which is a refinement of A_Y and for which R_P is acyclic.

6.2 Compatible Output Sets

In this section a criteria for partitioning algorithms is developed. Suppose that an algorithm contains two $R_Y^{\dagger}$ related outputs. By the path ordering property, this algorithm must be included at least twice in any output simulation sequence.

In the example problem of Section 2.3 outputs y_4 and y_5 are R_Y-related and are elements of algorithm F^2. If the algorithm F^2 is split into two algorithms F_1 and F_2, where

$$F_1 = F^2 - \{y_4\} = \{y_5\}$$

$$F_2 = F^2 - \{y_5\} = \{y_4\}$$

and a minimum-length simulation sequence is obtained for the new algorithm partition, this sequence will be a minimum-length simulation sequence.

In general there can be many pairs of $R_Y^{\dagger}$-related outputs. A <u>compatible output set</u> is a set of outputs containing no pair of $R_Y^{\dagger}$-related outputs. A singleton output set is a compatible output set for any acyclic output dependency relation. The reader will note that any partition representing a minimum-length output simulation sequence will contain only compatible sets. Thus if an algorithm is not a compatible set, only subsets of this algorithm need be considered in the formation of an output simulation sequence.

For each output algorithm F consider the <u>output algorithm graph</u> $<F,R_Y^{\dagger}>$. This graph represents the total dependency relation between outputs in the algorithm F.

The output algorithm graphs for the three algorithms of the example problem are shown in Figures 11a, 11b, and 11c. Only F^2 contains $R_Y^{\dagger}$-related outputs.

$\langle F^1, R_Y^\dagger \rangle$

$$\begin{tikzcd} y_1 \\ y_2 \\ y_3 \end{tikzcd}$$

(a) Output Algorithm Graph for F^1

$\langle F^2, R_Y^\dagger \rangle$

$$\begin{tikzcd} y_4 \arrow[r] & y_5 \end{tikzcd}$$

(b) Output Algorithm Graph for F^2

$\langle F^3, R_Y^\dagger \rangle$

$$\begin{tikzcd} y_6 & y_7 \end{tikzcd}$$

(c) Output Algorithm Graph for F^3

FIGURE 11

Output Algorithm Graphs for Algorithms of Figure 6

Theorem 6.3

The forward-level partition of $<F, R_Y^{\dagger}>$ is a minimum rank partition of F into compatible sets.

Proof

If two outputs are $R_Y^{\dagger}$ related they must be in separate levels. Each forward-level set is thus a compatible set. To prove minimum rank, suppose that the output algorithm graph contains a maximal-length path containing ℓ nodes. Clearly any compatible set can contain at most one of these nodes. The rank of the partition is thus greater than or equal to ℓ. The rank of the forward-level partition is equal to ℓ. Q.E.D.

In a similar manner it can be proved that the backward level set partition of $<F, R_Y^{\dagger}>$ is a minimum-rank partition of F into compatible sets. In fact, the forward- and backward level set partitions are only two of a family of minimum-rank level partitions. The choice of minimum-rank partition is relevant to the construction of a minimum-length simulation sequence.

6.3 Maximal Algorithm Subset

In this section a representation for the family of level partitions will be developed. The number of sets in a level partition has previously been shown to be equal to the number of nodes in the longest path. By the path ordering property,outputs on the same path must be in different level sets. Each output of a maximal-length path is in a separate level set and each level set contains one output of each maximal-length path.

For the purpose of discussion it will be assumed that the level sets are numbered in the order in which they will form a simulation sequence. The first forward level set is thus L_0, and the first backward-level set is $L^C_{\ell-1}$.

Suppose that an output is an isolated node, that is, it is not $R^{\dagger}_{Y}$ related to any other output in the algorithm. Clearly this output can be included in any level set. In a forward level partition isolated outputs are included in the first level set. In the backward-level partition isolated outputs are included in the last level set. In fact, the forward level partition has the property that each output is included in the earliest level set and the backward-level partition has the property that each output is included in the latest level set.

Consider the output algorithm graph of Figure 12, the forward and converse level partitions of this graph are:

$L_0 = \{1,3,8,9\}$	$L^C_5 = \{3\}$
$L_1 = \{2,4\}$	$L^C_4 = \{4\}$
$L_2 = \{6\}$	$L^C_3 = \{6,8\}$
$L_3 = \{7\}$	$L^C_2 = \{7\}$
$L_4 = \{5\}$	$L^C_1 = \{1,5\}$
$L_5 = \{10\}$	$L^C_0 = \{2,9,10\}$

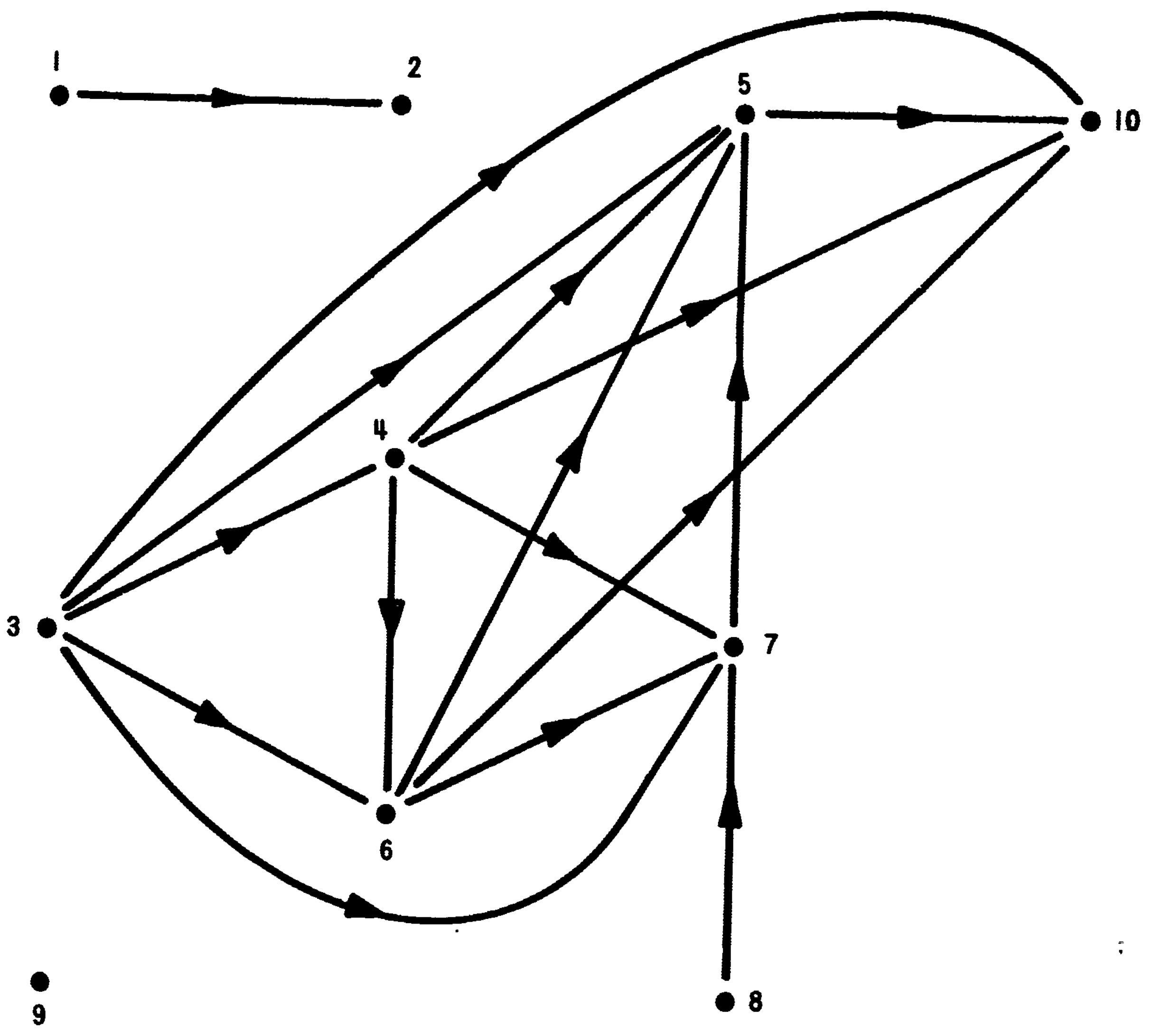


FIGURE 12

OUTPUT ALGORITHM GRAPH

This output algorithm graph has a path containing six nodes, i.e., the path

$$3 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 5 \rightarrow 10$$

Hence the algorithm must be included at least six times in any output simulation sequence. After the first application of this algorithm only nodes in L_0 can achieve a "1" simulation state. In the initial application of this algorithm only nodes in L_0 need be considered. If this algorithm is to be included only six times in the output simulation sequence, the nodes of L_0 which are on a maximal-length path must achieve "1" simulation state after L_0. These nodes are also in $L^C_{\ell-1}$. In the output algorithm graph of Figure 12 only node 3 must achieve a "1" simulation state after the first application of the algorithm. Nodes 1, 8, and 9 can achieve a "1" simulation state after a later application. These nodes can achieve a "1" simulation state after the second application. Additional nodes which can achieve a "1" simulation state after the second application are the nodes in L_1. After the second application of the algorithm in any sequence containing only a minimum number of these algorithms, the set of nodes which can attain a "1" simulation state is $(L_0 - L^C_{\ell-1}) \cup L_1$. Only nodes in the above set need be considered in the second application.

In general the set of outputs which can attain a "1" simulation state after the nth application of an algorithm in any sequence containing only a minimum number of these algorithms is called a maximal subset of an algorithm. A maximal subset is not necessarily a compatible set. For the output algorithm graph of Figure 12

$$\left(L_0 - L^C_{\ell-1}\right) \cup L_1 = \{1,2,4,8,9\}$$

This set is not a compatible set since nodes 1 and 2 are R_Y related. Outputs 1 and 2 are on a path containing two nodes. Since at least six applications of the algorithms are required, any one of the first five applications may set the simulation state of output 1 equal to "1".

Given the forward and backward-level partitions the family C_i of maximal subsets is formed as follows:

$$C_0 = L_0$$

$$C_1 = \left(C_0 - L^C_{\ell-1}\right) \cup L_1$$

$$\vdots$$

$$C_i = \left(C_{i-1} - L^C_{\ell-i}\right) \cup L_i$$

$$\vdots$$

For the acyclic output algorithm graph of Figure 12, the maximal subsets are:

$$C_0 = \{1,3,8,9\}$$

$$C_1 = \{1,2,4,8,9\}$$

$$C_2 = \{1,2,6,8,9\}$$

$$C_3 = \{1,2,7,9\}$$

$$C_4 = \{1,2,5,9\}$$

$$C_5 = \{2,9,10\}$$

The family of maximal algorithm subsets is a cover of the family of minimum-rank level partitions, that is, each forward level set L_i is a subset of the corresponding maximal subset C_i and each backward level set L_i^C is a subset of $C_{\ell-i-1}$.

6.4 Refinement of Maximal Algorithm Subsets

In Section 6.3, a family of maximal subsets was defined for each algorithm. These subsets are a cover of the family of minimum-rank level partitions. In the maximal subsets only nodes on a maximal-length path in the output algorithm graph are included in a single unique subset. In this section refinements of maximal algorithm subsets are developed.

An output of an algorithm is ordered if it is included in exactly one maximal subset. The output algorithms graph of Figure 12 contains one maximal-length path with the following six nodes:

$$3 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 5 \rightarrow 10$$

The above nodes are ordered. The remaining nodes are included in more than one maximal subset and hence not ordered.

In the formation of maximal algorithm subsets only the dependency of outputs on outputs in the same algorithm was considered. In this section the structure of the network output graph is examined.

The predecessor set of an output y, written P(y), is the set of outputs which have a path to output y.

$$P(y) = \left\{ y' \mid y' R_Y^{\dagger} y \right\}$$

The successor set of an output y, written S(y), is the set of outputs which are accessible from output y.

$$S(y) = \left\{ y' \mid y R_Y^{\dagger} y' \right\}$$

Consider the network output graph of Figure 13. The output algorithm graph of algorithm F^3 was shown in Figure 12. The maximal algorithm subsets were developed in the previous section. Output 8 is included in the first three subsets.

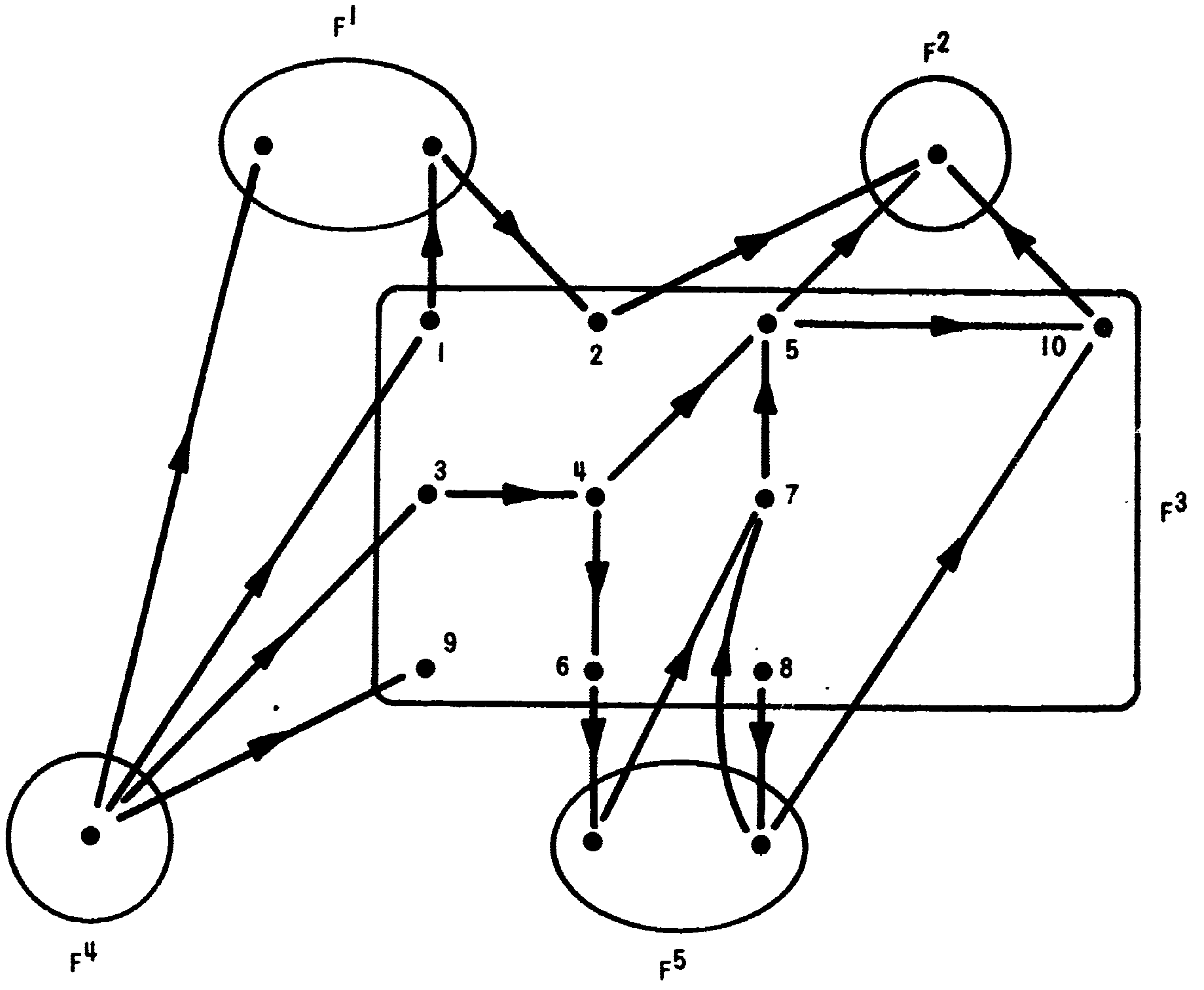


FIGURE 13

NETWORK OUTPUT GRAPH

Since it has no predecessors, the first application of algorithm F^3 will set its simulation state equal to "1". Output 8 thus need be included only in C_0.

Rule 1

If the predecessor set of an output is empty, include this output only in C_0.

Next consider output 9. This output has no successors. Thus no output depends upon this output. In this case it is desirable to include output 9 in the last maximal subset.

Rule 2

If the successor set of an output is empty, include this output only in the last subset.

Output 1 has the same predecessor set as output 3. Since outputs 1 and 3 are in the same algorithm, any algorithm sequence which sets the simulation state of one output equal to "1" will also set the simulation state of the other output equal to "1". These outputs can thus be included in the same subset. Output 3 is ordered and included only in C_0. Output 1 can thus be ordered.

Rule 3

If output y is included only in subset n and $P(y) = P(y')$ include output y' only in subset n.

Output 2 and output 10 have the same successor set. All outputs which are dependent upon output 2 are also dependent upon output 10 and vice versa. Outputs 2 and 10 can thus be included in the same subset.

Rule 4

If output y is included only in subset n and $S(y) = S(y')$ include output y' only in subset n.

The ordering rules presented here are frequently contradictory. For example, rule 2 can be applied to include output 9 into subset C_5 or rule 3 can be applied to include output 9 into subset C_0. Either application can lead to the formation of a minimum-length output simulation sequence.

For the network output graph of Figure 13, the refined maximal algorithm subsets are:

$$C_0 = \{1,3,8\}$$

$$C_1 = \{4\}$$

$$C_2 = \{6\}$$

$$C_3 = \{7\}$$

$$C_4 = \{5\}$$

$$C_5 = \{2,9,10\}$$

The network output graph with the above subsets substituted for F^3 is shown in Figure 14. The algorithm dependency relation for this graph is acyclic. The sequence

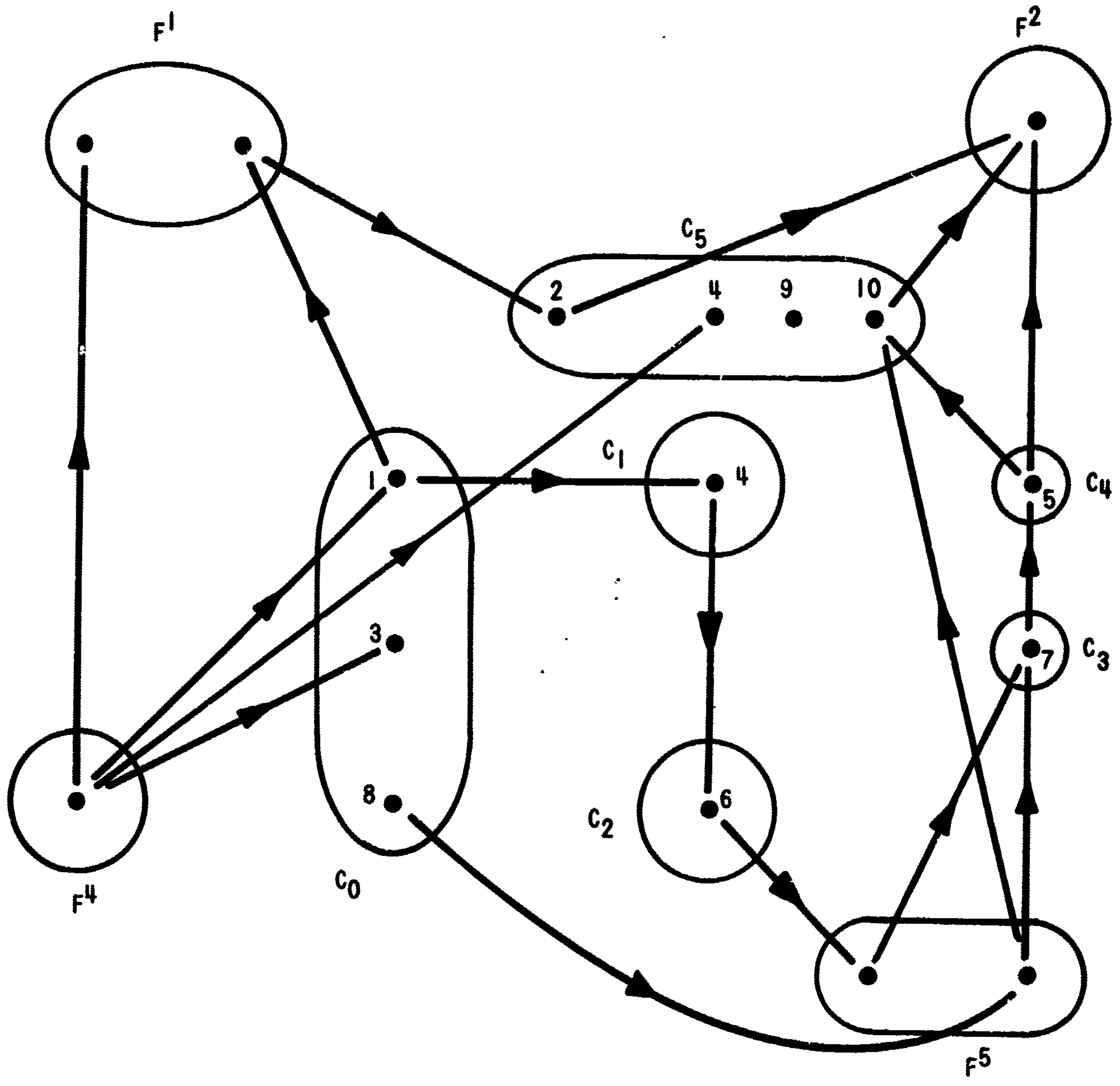


FIGURE 14

NETWORK OUTPUT GRAPH WITH MAXIMAL SUBSETS SUBSTITUTED FOR F^3

$$F^4F^3F^1F^3F^3F^5F^3F^3F^3F^2$$

is a minimum-length output simulation sequence.

In this example problem the algorithm splitting technique has resulted in an acyclic algorithm graph. This is not always the case. In the next chapter a general method for forming simulation sequences is presented.

VII. FORMATION OF SIMULATION SEQUENCES

In previous chapters various techniques for constructing simulation sequences were presented. In Chapter III the problem of constructing a simulation sequence was reduced to the problem of forming an output simulation sequence. In Chapter V an algorithm ordering relation was discussed and the simulation-sequence formation problem was reduced to a collection of smaller problems. A criterion for splitting algorithms was developed in Chapter VI. In this chapter these techniques are combined into a general method for constructing a minimum-length simulation sequence.

In Section 7.1 a selective search technique is presented. The general method is presented in Section 7.2 and an illustrative example is presented in Section 7.3.

7.1 Modified Eligible-Set Method

In this section a selective search technique for the construction of a simulation sequence will be developed. Although this technique is applicable to all acyclic networks, it is primarily intended for a problem to which none of the techniques discussed in Chapters V and VI apply, that is, problems represented by a connected algorithm graph of maximal algorithm subsets.

In Section 2.3 a simulation sequence was constructed via a sequential selection of algorithms. The choice of algorithms was restricted by a one-step look-ahead rule.

The eligible output set E_Y includes those outputs which do not have "1" simulation state but which can achieve "1" simulation state via a single application of an algorithm. Formally

$$E_Y = \{y | s_y \neq 1 \text{ and } y' R_Y y \Rightarrow s_{y'} = 1\}$$

The eligible algorithm set E_A is the set of algorithms which have at least one output in the eligible output set. Formally

$$E_A = \{F | y \in E_Y \text{ for some } y \in F\}$$

Both eligible sets are a function of the algorithm sequence previously applied. In the search for simulation sequences only algorithms in E_A need be considered. A minimum-length simulation sequence can be constructed by considering all eligible algorithms at each step. This method will require the construction of a large number of algorithm sequences. The computation time can be reduced drastically by recognizing optimal choices. Suppose that the entire set of outputs of an algorithm is included in the eligible output set. Selecting this algorithm as the next algorithm in the sequence is an optimal choice since the algorithm will be included only once in the simulation sequence. Each output in the algorithm will achieve a "1" simulation state and outputs which are dependent upon outputs in this algorithm may become eligible.

The done output set D_Y is the set of outputs which have "1" simulation state. The done algorithm set D_A is the set of algorithms which have all their outputs in the done output set. A selection of an algorithm is an optimal choice if the selection adds this algorithm to the done algorithm set. The set of maximum algorithms E_M includes those algorithms in E_A which, if selected, will become elements of D_A. **The order of choice of an algorithm from a set of maximum algorithms is immaterial.**

If the eligible algorithm set includes only a single algorithm, the selection of this algorithm is by default an optimal choice. In the case of two or more algorithms, each choice must be considered. This can be accomplished by a tree structure search. The flow chart displayed in Figure 15 employs such a technique.

The parameter L indicates the level of the tree structure. Initially L is equal to 1. L is increased by 1 whenever a nonoptimal choice is encountered. L is decreased by 1 after all choices in the largest level have been exausted.

Before incrementing L, a test is performed to check whether the current algorithm sequence can form a simulation sequence of length less the shortest simulation sequence found. The potential length of a simulation sequence is equal to the number of algorithms in the sequence plus the number

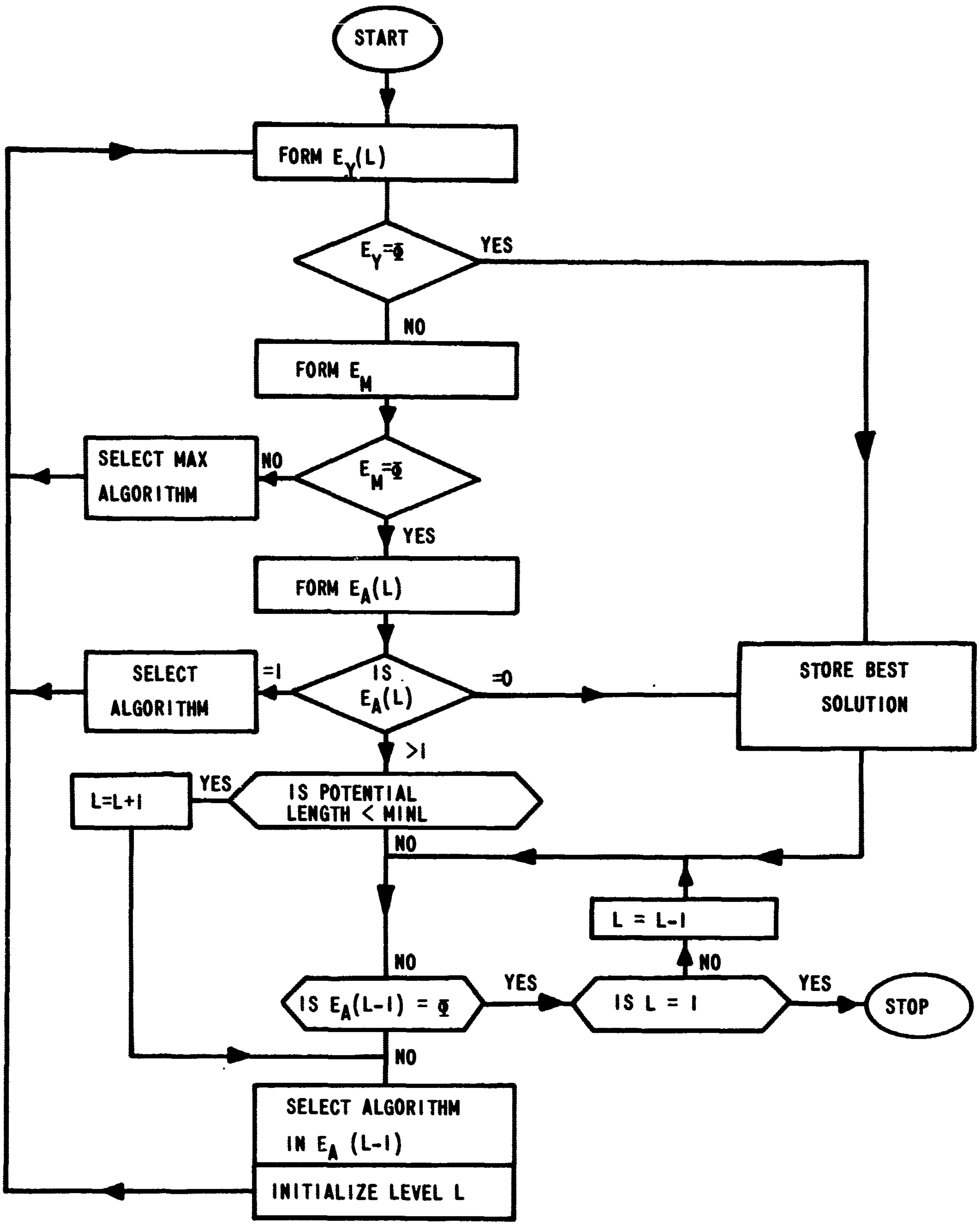


FIGURE 15

FLOW CHART FOR THE SELECTIVE SEARCH METHOD of algorithms which are not in D_A. If the potential length is greater than or equal to MINL, this sequence cannot form a minimum-length simulation sequence. In this case the routine abandons this sequence and selects the next choice in the highest level. If no choices remain and L is equal to 1 the routine stops. If L is greater than 1, L is decreased by 1.

Whenever an algorithm is selected, this algorithm is removed from the eligible algorithm set and added to the simulation sequence. The outputs in the algorithm which achieve "1" simulation state are added to the done output set. A new eligible set is formed and the procedure is continued.

If the eligible output set or the eligible algorithm set is empty, all outputs which can achieve "1" simulation state have achieved "1" simulation state and the length of the sequence formed is compared with the shortest sequence found. The reader will note that this sequence is a simulation for any acyclic network. In the case of a cyclic network the sequence formed will result with a "1" simulation state for all outputs which are not dependent upon outputs in a cycle. The modified eligible set method thus forms a simulation sequence for acyclic networks and in the case of cyclic networks this method will form a sequence which will result in "1" simulation state for all outputs which can achieve "1" simulation state.

Consider the example problem discussed in Section 2.3. The network output graph for this problem was shown in Figure 5, the algorithm graph was given in Figure 6, and the output algorithm graphs were given in Figure 11. Algorithm F^2 is split into two algorithms by the algorithm splitting technique. The network output graph with maximal subsets substituted for F^2 is shown in Figure 16. In order to simplify the notation each node has been labeled with the output subscript.

Initially L = 1, MINL = 7 (number of outputs), $SEQ(1) = \lambda$, $D_Y(1) = \Phi$ and $D_A(1) = \Phi$. The eligible sets are

$$E_Y = \{2,3,4\}, \quad \text{and} \quad E_M = \{F_2\}.$$

Since E_M is not empty, the algorithm in E_M is selected

$$SEQ(1) = F_2, \quad D_Y(1) = \{4\} \text{ and } D_A(1) = \{F_2\}.$$

The eligible sets become:

$$E_Y = \{2,3,6\}, \quad E_M = \Phi \quad \text{and} \quad E_A(1) = \{F^1, F^3\}.$$

A choice exists. The potential length, 4, is less than 7. Thus consider each choice, set L = 2, select F^1, and initialize the new level.

$$SEQ(2) = F^1, \quad D_Y(2) = \{2,3,4\} \quad \text{and} \quad D_A(2) = \{F_2\}.$$

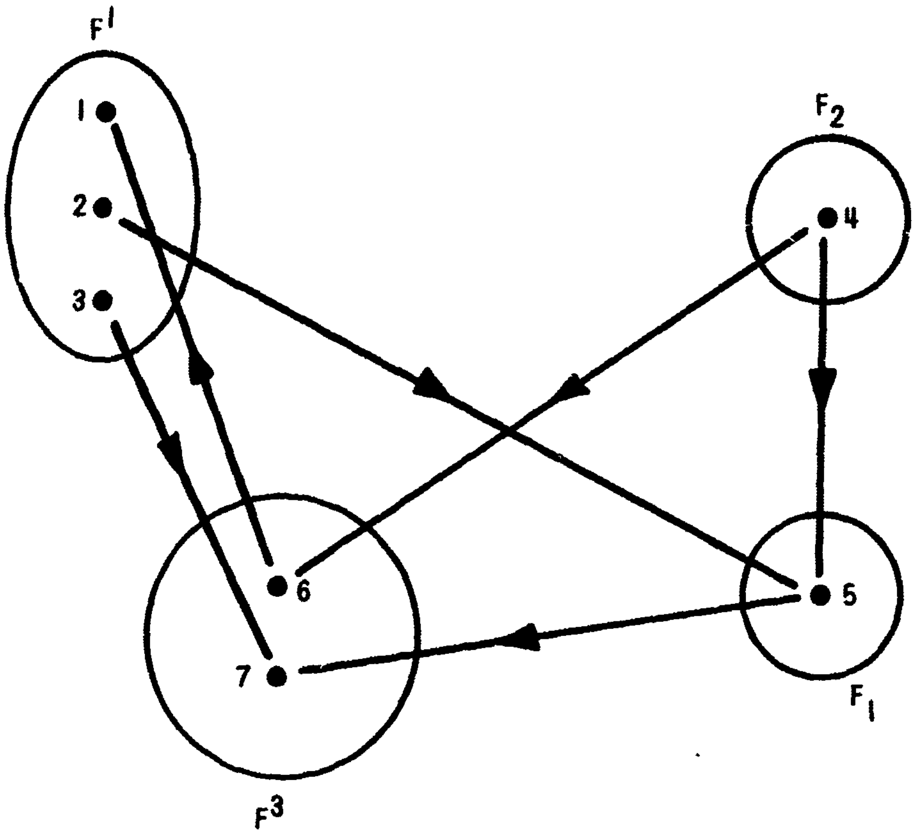


FIGURE 16

NETWORK OUTPUT GRAPH OF FIGURE 6b
WITH MAXIMAL SUBSETS SUBSTITUTED FOR F^2

The eligible sets are now

$$E_Y = \{5,6\} \quad \text{and} \quad E_M = \{F_1\}.$$

E_M is non-empty, thus select F_1.

$$SEQ(2) = F^1F_1, \quad D_Y(2) = \{2,3,4,5\} \text{ and } D_A(2) = \{F_1,F_2\}.$$

The new eligible sets are

$$E_Y = \{6,7\} \quad \text{and} \quad E_M = \{F^3\}$$

F^3 is a maximum algorithm.

$$SEQ(2) = F^1F_1F^3, \quad D_Y(2) = \{2,3,4,5,6,7\}, \quad D_A(2) = \{F_1,F_2,F^3\},$$

$$E_Y = \{1\}, \quad E_M = \{F^1\}.$$

F^1 is a maximum algorithm since only output 1 remains to be done.

$$SEQ(2) = F^1F_1F^3F^1, \quad D_Y(2) = \{1,2,3,4,5,6,7\}, \quad D_A(2) = \{F^1,F_1,F_2,F^3\}$$

and $E_Y = \Phi$.

The sequence $F_2F^1F_1F^3F^1$ is a simulation sequence.

$$MINL = 5$$

The routine now selects F^3, initializes level 2 and proceeds as follows:

$$SEQ(2) = F^3 \quad D_Y(2) = \{4,6\} \quad D_A(2) = \{F_2\}$$

$$E_Y = \{1,2,3\} \quad E_M = \{F^1\}$$

$$SEQ(2) = F^3F^1 \quad D_Y(2) = \{1,2,3,4,6\} \quad D_A(2) = \{F^1,F_2\}$$

$$E_Y = \{5\} \quad E_M = \{F_1\}$$

$$SEQ(2) = F^3F^1F_1 \quad D_Y(2) = \{1,2,3,4,5,6\} \quad D_A(2) = \{F^1,F_1,F_2\}$$

$$E_Y = \{7\} \quad E_M = \{F^3\}$$

$$SEQ(2) = F^3F^1F_1F^3 \quad D_Y(2) = \{1,2,3,4,5,6,7\}$$

$$D_A(2) = \{F^1,F_1,F_2,F^3\}$$

$$E_Y = \Phi$$

The new simulation sequence is $F_2F^3F^1F_1F^3$. In this case no choices remain and the routine terminates.

The eligible set algorithm can form a minimum-length simulation sequence for any network with acyclic R_Y. The algorithm forms the simulation sequence by a sequential selection of algorithms. A one-step look ahead technique is employed. Optimal choices are recognized and a potential-length test abandons simulation sequences whose length will be greater than the minimum-length.

The amount of computation that is required is dependent upon the problem. If the algorithm graph is acyclic, the maximum algorithm set is never empty and the number of eligible sets formed is equal to the number of nodes in the longest path of the algorithm graph. For the network of Figure 7, the routine will select F^1 in the first step, F^2 and F^4 in the second step, and F^3 in the third and final step. The routine is thus very efficient for acyclic algorithm graphs.

In the case of a cyclic algorithm graph the amount of computation is dependent upon the choice of algorithms. The routine will examine all potential minimum-length simulation sequences. In the routine a potential minimum-length simulation sequence is any simulation sequence which is shorter than the shortest sequence found. Although the routine will halt with a minimum-length sequence, the amount of computation is dependent upon the length of sequences found. If the initial selection results in a short sequence, a large number of sequences will be abandoned by the potential length test.

The amount of computation can be reduced by incorporating simple selection rules which by ordering eligible algorithms will cause the search technique to examine the more likely sequences first. Prior to any selection of a nonmaximum algorithm each eligible algorithm is examined and the best algorithm is selected. The following selection rules have been investigated by the author:

1. Select the algorithm which will update the most outputs.
2. Update the output in the lowest forward level.
3. Update the output in the highest backward level.

None of these rules apply to all problems. The author has constructed problems to which none of the above rules apply. Rule 1 is based on the presumption that if the maximum number of outputs is updated at each step, the number or steps will be minimized. The presumption is false and Rule 1 is not a good selection criteria since output fan-outs are not considered. The reader will note that if an output does not fan out, then whether this output is updated at the present step or at a future step is immaterial.

Rules 2 and 3 are based on the path criterion, that is, outputs that are initial nodes of paths in the network graph must be updated before their successors can be updated. The level partitions of the network output graph specify such an ordering. Rule 3 is usually a better selection rule than Rule 2 since in Rule 3 outputs that are initial nodes of maximal-length paths are given preference.

The rules mentioned above are simple examples of selection rules. More powerful selection rules can be developed. The implementation of selection rules will reduce the amount of computation only if the computation required by the selection rule is less than the computation required by the eliminated sequences. It is the opinion of the author that except for special uses the implementation of selection rules will not significantly reduce the total amount of computation.

7.2 General Method

Although the modified eligible-set method presented in the previous section can form minimum-length simulation sequences for any acyclic network, it does not employ the maximal connected subset partitioning technique developed in Chapter V. In this section a general method is presented. This method is simply an ordering of the techniques previously discussed.

1. Calculate maximum algorithm subsets.
2. Form an algorithm graph using the maximum algorithm subsets.
3. Partition the algorithm graph into maximal-connected subgraphs.
4. Level order the maximal-connected subsets.
5. Obtain a minimum-length simulation sequence for each maximal-connected subset.

In the general method the algorithm splitting technique discussed in Chapter VI is applied first. This technique results in a new set of algorithms. The partitioning of the ordering problem discussed in Chapter V is applied next and the modified eligible set method is applied last to each maximal connected subset.

A computer program has been written in the FORTRAN V language. This program forms simulation sequences

using the general method. The program contains approximately 1400 cards and is operational on the Univac 1108 computer.*
The operation of the program in the solution of an example problem is described in the next section.

7.3 Illustrative Problem

In this section a simulation sequence will be constructed for an illustrative problem. The system to be simulated is composed of 12 subsystems. Each subsystem contains memory and at least two output variables. The process variable dependency relation is assumed to be known for each process. The network interconnections are also known. The behavior of each process is described by two algorithms, one algorithm calculates new values for the state variables and a second algorithm calculates new values for the output variables.

The output dependency relation for this network is shown in Figure 17. The outputs and processes have been numbered numerically. The algorithm definition is stored by the computer program in the matrix ALG. This matrix is shown in Figure 18. The output dependency relation is stored in the Boolean matrix OUT. The entry in row i and column j of this matrix is equal to 1 if output j directly depends upon output i. The OUT matrix is shown in Figure 19. The row 2, column 5 entry of this matrix is 1 since output 5 depends upon output 2.

* A listing of the program is available from the author. A user bulletin for this program is presented in the appendix.

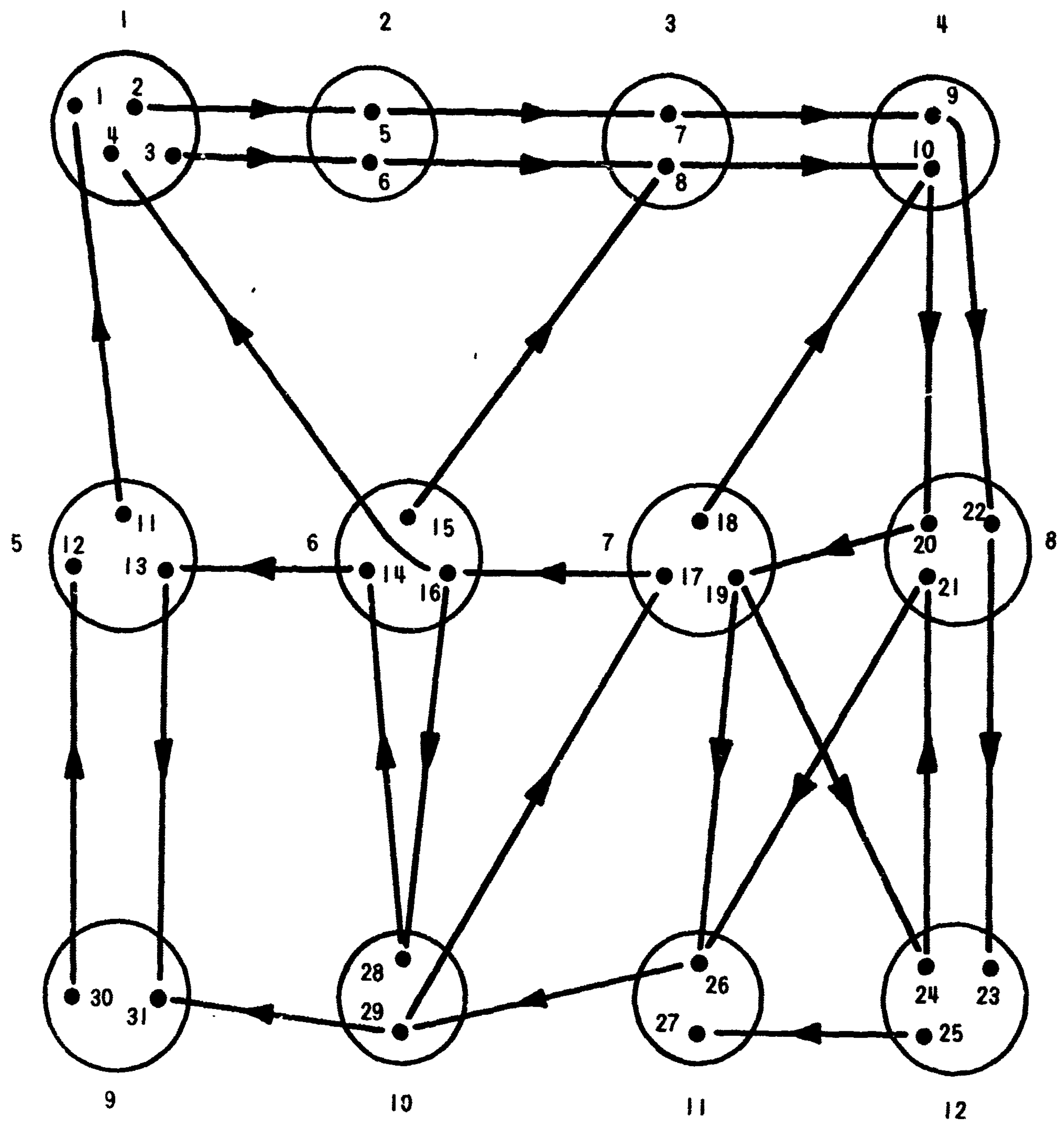


FIGURE 17

NETWORK OUTPUT GRAPH

ROW NO.	ROW ENTRIES				MATRIX ALG
1	1	2	3	4	
2	5	6			
3	7	8			
4	9	10			
5	11	12	13		
6	14	15	16		
7	17	18	19		
8	20	21	22		
9	30	31			
10	28	29			
11	26	27			
12	23	24	25		

FIGURE 18

Algorithm Definition

```
                           MATRIX OUT

                       1         2         3
ROW           1234567890123456789012345678901

  1           0000000000000000000000000000000
  2           0000100000000000000000000000000
  3           0000010000000000000000000000000
  4           0000000000000000000000000000000
  5           0000001000000000000000000000000
  6           0000000100000000000000000000000
  7           0000000010000000000000000000000
  8           0000000001000000000000000000000
  9           0000000000000000000001000000000
 10           0000000000000000000100000000000
  1           1000000000000000000000000000000
  2           0000000000000000000000000000000
  3           0000000000000000000000000000001
  4           0000000000001000000000000000000
  5           0000000100000000000000000000000
  6           0001000000000000000000000001000
  7           0000000000000001000000000000000
  8           0000000001000000000000000000000
  9           0000000000000000000000010100000
 20           0000000000000000001000000000000
  1           0000000000000000000000000100000
  2           0000000000000000000000100000000
  3           0000000000000000000000000000000
  4           00000000000C0000000010000000000
  5           0000000000000000000000000010000
  6           0000000000000000000000000000100
  7           0000000000000000000000000000000
  8           0000000000000100000000000000000
  9           0000000000000000100000000000001
 30           0000000000010000000000000000000
  1           0000000000000000000000000000000
```

FIGURE 19

Output Dependency Relation Matrix

Each entry in this matrix represents one bit of a computer memory word. Since the Univac 1108 computer has 36 bits per word, each row of the matrix requires only a single computer word. Besides reducing the storage requirement, packing speeds up row operations since logical operations between corresponding bits can be performed simultaneously.

Many graphical operations can be reduced to a succession of row operations. For example a fast routine for forming the transitive closure of a relation represented by a Boolean matrix due to Warshall [10] is the following:

> Scan the Boolean matrix using a vertical raster scan. Whenever a 1 is encountered in row i column j, replace row i by the logical inclusive OR function of row i and row j.

This routine applied to the matrix of Figure 19 results in the matrix of Figure 20. The matrix SUC represents the transitive closure of matrix OUT.

The program uses the matrix SUC to calculate the maximum algorithm subsets. The subroutine, which forms maximum algorithm subsets, first calculates the condensation of matrix SUC with respect to matrix ALG. If a diagonal element of the resulting matrix is equal to 1, the algorithm corresponding to the diagonal element is not a compatible set. A submatrix of SUC containing only rows and columns corresponding to outputs in this algorithm is formed. The maximal subsets are calculated from the forward and converse level partitions of the subgraph and the refined maximal

```
                     MATRIX SOC

                          1         2         3
ROW        1234567890123456789012345678901

  1        0000000000000000000000000000000
  2        0001010100000000000001100000000
  3        0001010101001101101110010101101
  4        0000000000000000000000000000000
  5        0000001010000000000001100000000
  6        0001000101001101101110010101101
  7        0000000010000000000001100000000
  8        0001000001001101101110010101101
  9        0000000000000000000001100000000
 10        0001000000001101101110010101101
  1        1000000000000000000000000000000
  2        0000000000000000000000000000000
  3        0000000000000000000000000000001
  4        0000000000001000000000000000001
  5        0001000101001101101110010101101
  6        0001000000001100000000000001001
  7        0001000000001101000000000001001
  8        0001000001001101101110010101101
  9        0001000000001101100010010101101
 20        0001000000001101101010010101101
  1        0001000000001101100000000101101
  2        0000000000000000000000100000000
  3        0000000000000000000000000000000
  4        0001000000001101100010000101101
  5        0000000000000000000000000010000
  6        0001000000001101100000000001101
  7        0000000000000000000000000000000
  8        0000000000001100000000000000001
  9        0001000000001101100000000001001
 30        0000000000001000000000000000000
  1        0000000000000000000000000000000
```

FIGURE 20

Total Output Dependency Relation Matrix

subsets are substituted for the original algorithm definition. The resulting maximum subset definitions are shown in Figure 21. These splits are displayed in Figure 22. The reader will note that the subroutine was unable to refine the split of algorithm 8. Output 22 is in maximum algorithm number 8 and maximum algorithm number 18.

The condensation of the network relation with respect to the maximal subsets is represented by the Boolean matrix of Figure 23. This matrix represents the maximal subset output algorithm graph. The maximal connected subsets of this graph are constructed by the following routine:

1. Form the transitive closure of the algorithm dependency relation.
2. Obtain the symmetric portion of this matrix.
3. Set diagonal elements equal to 1.

Each distinct row of the resulting matrix is a maximal connected subset. The maximal connected algorithm subsets for the example problem are shown in Figure 24 and the dependency relation among maximal connected algorithm subsets is represented by the Boolean matrix of Figure 25. The converse level partition of the subset dependency relation of Figure 25 is shown in Figure 26. According to this partition maximal connected algorithm subset 11 occurs first in the simulation sequence. This subset includes only maximal algorithm 13 which is a subset of algorithm 1. The first algorithm in the simulation sequence is thus algorithm 1.

ROW NO.	ROW ENTRIES			MATRIX MAX AL
1	1	4		
2	5	6		
3	7	8		
4	9	10		
5	11	12	13	
6	14			
7	17			
8	21	22		
9	30	31		
10	28			
11	26	27		
12	23	24	25	
13	2	3		
14	16			
15	15			
16	19			
17	18			
18	20	22		
19	29			

FIGURE 21

Maximum Algorithm Definition

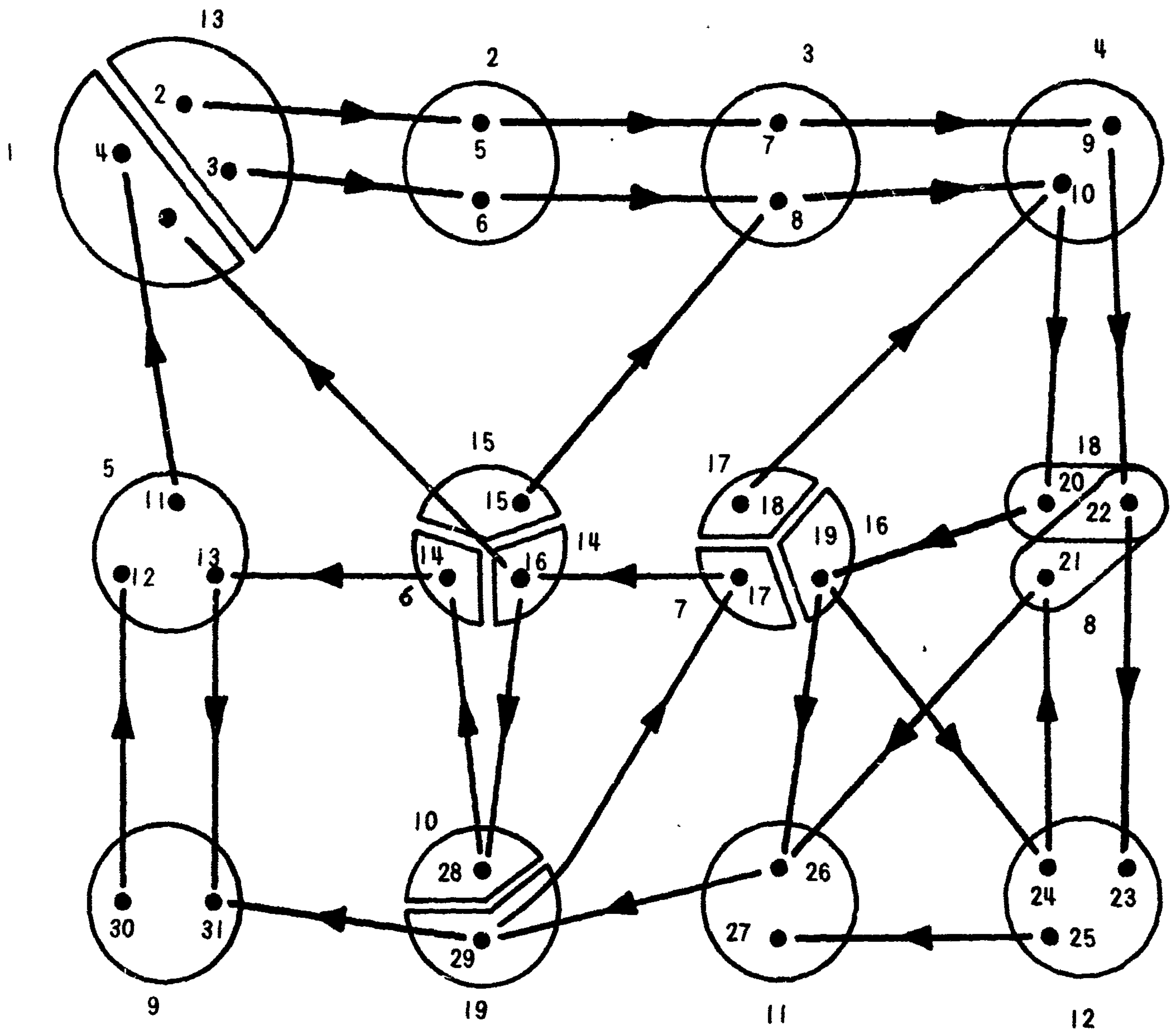


FIGURE 22

NETWORK OUTPUT GRAPH WITH

MAXIMAL ALGORITHM SUBSETS

```
                    MATRIX M AL G

                    1
ROW       1234567890123456789

 1        0000000000000000000
 2        0010000000000000000
 3        0001000000000000000
 4        0000000100000000010
 5        1000000010000000000
 6        0000100000000000000
 7        0000000000000100000
 8        0000000000110000000
 9        0000100000000000000
10        0000010000000000000
 1        0000000000000000001
 2        0000001001000000000
 3        0100000000000000000
 4        1000000001000000000
 5        0010000000000000000
 6        0000000000110000000
 7        0001000000000000000
 8        0000000000010001000
 9        0000001010000000000
```

FIGURE 23

Maximal Subset Algorithm Graph Matrix

ROW NO.	ROW ENTRIES		MATRIX C.A.P
1	1		
2	2		
3	3		
4	4		
5	5	9	
6	6		
7	7		
8	8	12	
9	10		
10	11		
11	13		
12	14		
13	15		
14	16		
15	17		
16	18		
17	19		

FIGURE 24

Maximal Connected Algorithm Subsets

```
                          MATRIX CAGS G

                  1
ROW      12345678901234567

  1      00000000000000000
  2      00100000000000000
  3      00010000000000000
  4      00000001000000010
  5      10000000000000000
  6      00001000000000000
  7      00000000000100000
  8      00000000010000000
  9      0000010000C000000
 10      00000000000000001
  1      01000000000000000
  2      10000000100000000
  3      00100000000000000
  4      00000001010000000
  5      00010000000000000
  6      00000001000001000
  7      00001010000000000
```

FIGURE 25

Maximal Connected Algorithm Subset

Dependency Relation Matrix

ROW NO.	ROW ENTRIES		MATRIX C.A.OR
1	1		
2	5		
3	6		
4	9		
5	12		
6	7		
7	17		
8	10		
9	8		
10	14		
11	16		
12	4		
13	3	15	
14	2	13	
15	11		

FIGURE 26

Converse Level Partition of

the Maximal Connected Algorithm

Subset Graph

The second and third connected algorithm subsets are 2 and 13. These subsets correspond to algorithms 2 and 6. The subsets 3 and 15 correspond to algorithms 3 and 7. Subsets 4, 16 and 14 correspond to algorithms 4, 8 and 7 respectively. Subset 8 contains maximal algorithms 8 and 12. Maximal algorithm 8 contains outputs 21 and 22 while maximal algorithm 12 contains outputs 23, 24 and 25. Output 22 has "1" simulation state since it was updated by a prior application of algorithm 8. With this initial condition the modified eligible set method forms the simulation sequence 12 8 for subset 8.

Except for subset 5 the remaining subsets correspond to single algorithms. The modified eligible set method forms the sequence 5 9 5 for subset 5. The resulting simulation sequence is

1 2 6 3 7 4 8 7 12 8 11 10 7 6 10 6 5 9 5 1.

The only searching required to obtain the above solution was in the construction of a simulation sequence for subset 5 and only two sequences were examined in the construction of a simulation sequence for this subset.

The modified eligible set method was used to calculate a simulation sequence for this network without the aid of algorithm splitting. This method used a maximum of 12 levels. The number of choices at each level varied from 2 to 6. Approximately 500,000 potential sequences were examined. With the implementation of the algorithm splitting and partitioning techniques the problem became trivial.

VIII. EXTENSIONS

In the previous chapters a single characterization of the simulation sequence ordering problem was employed. In this chapter various extensions of the problem are discussed.

In the system model, it was assumed that each process contained a single algorithm which calculated new values for the set of state variables. This assumption greatly simplified the construction of simulation sequences. In Section 8.1 the single algorithm per set of state variables assumption is replaced by a single algorithm per process assumption.

In the previous chapters it was assumed that a minimum-length simulation sequence was desired. Actually the simulation sequence with minimum cost is usually desired. In Section 8.2 each algorithm is assumed to have a positive weight associated with it and the formation of a simulation sequence with minimum linear sum is discussed.

In Section 8.3, the modelling of random variables is discussed. The problem of forming a general simulation sequence for a network with several configurations is discussed in Section 8.4. In Section 8.5 the elimination of time steps is discussed.

8.1 Single Algorithm Characterization

In Chapter II in order to guarantee that a simulation sequence will exist for any process it was assumed that a single algorithm calculated new values for the set of state variables. In this section it will be assumed that a single algorithm exists for each process. This algorithm will calculate new values for all process output and state variables. This is the characterization employed by Parnas [10]. Parnas considers two types of application of a process algorithm, H^p. In the first type of application, denoted by H^p, new values are obtained for all process p output and state variables. In the second type of application, denoted by $\sim H^p$, new values are obtained for all process p output and state variables, but the state variable values are restored to their initial value after the application of the algorithm. An application of $\sim H^p$ thus effectively calculates new values only for output variables.

Parnas assumes that only the input-output process dependency relation is known. Parnas [10] states: "Correct values for the state variables can be determined if and only if the event description algorithm is executed with all input variables known to be correct." He therefore assumes that each process state variable is dependent upon all the input and state variables of this process.

Using this characterization a minimum-length simulation sequence can be obtained as follows:

1. Add to each process an output y^*. The value of y^* is assumed to be dependent upon all process inputs. Output y^* is not R_N related.

2. Assume that for each process there exists a single algorithm which will calculate new values for each process output (including y^*).
3. Obtain a minimum-length output simulation sequence.
4. For each process p, scan the output simulation sequence for occurrences of the process p output algorithm. Label the last occurrence H^p and all preceding occurrences $\sim H^p$.

The resulting sequence is a minimum-length simulation sequence for the single algorithm characterization.

Consider the example problem discussed in Section 2.3. The network for this example is displayed in Figure 4. The network output graph is displayed in Figure 27. A minimum-length output simulation sequence is $F^2F^3F^1F^2F^3$. A minimum-length simulation sequence for the single algorithm characterization is thus $\sim H^2 \sim H^3 H^1 H^2 H^3$.

8.2 Weighted Algorithms

In the previous chapters it has been assumed that a minimum-length simulation sequence is desired. If the cost of each algorithm is known, a simulation sequence with <u>minimum cost</u> is usually desired. The problem of constructing such a sequence is discussed in this section. Each algorithm is assumed to have a positive weight associated with it and the simulation sequence with minimum linear sum is desired.

The approach employed in the formation of a minimum-length simulation sequence is to include each algorithm a minimum number of times in the simulation sequence. Since each algorithm has positive weight, such a sequence is a minimum-weight simulation sequence for any positive

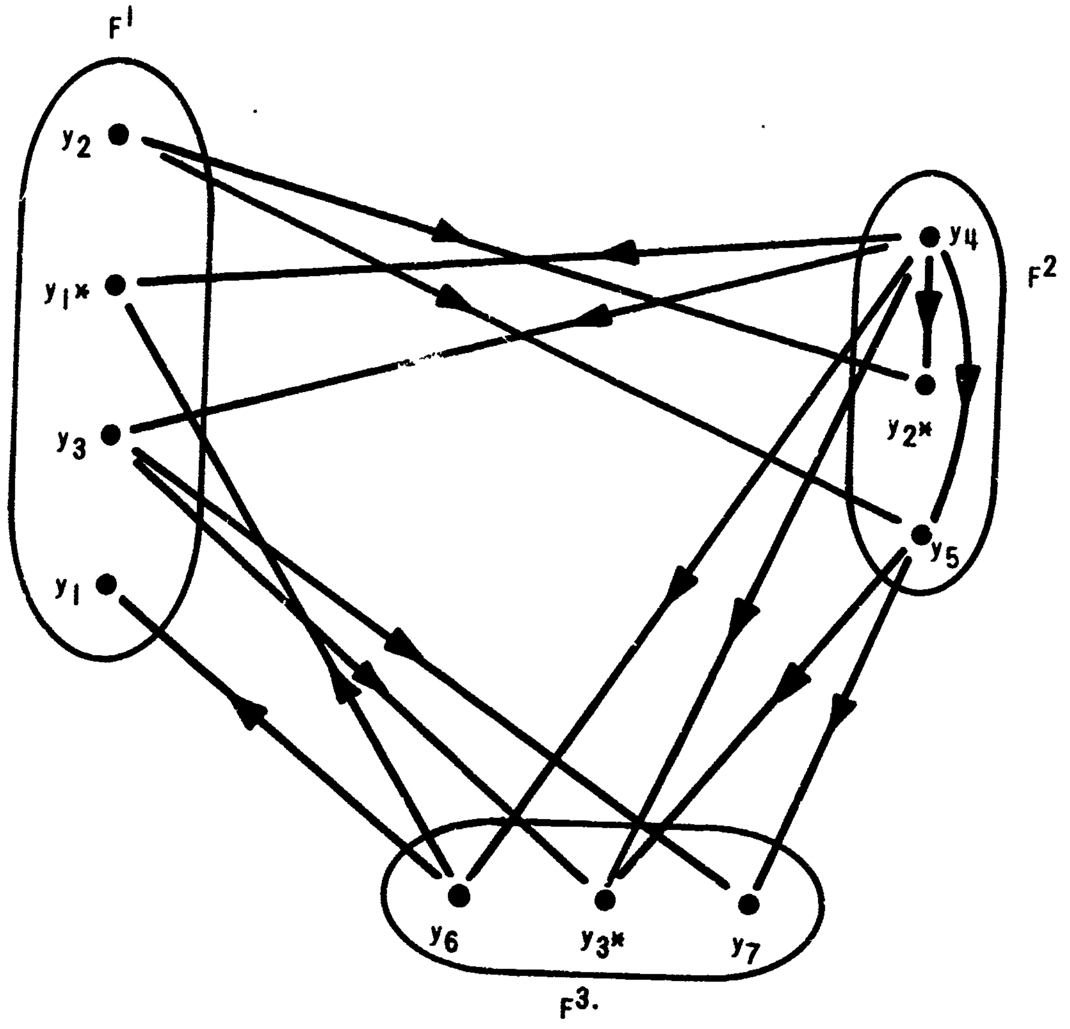


FIGURE 27

MODIFIED NETWORK OUTPUT GRAPH

weight assignment. The algorithm splitting technique discussed in Chapter VI is thus applicable to this problem.

In Chapter V a forward level partition of the algorithm graph was shown to form a minimum-length simulation sequence. In this sequence each algorithm is included exactly once. This sequence is also a minimum-weight sequence for any positive weight assignment. Furthermore, in the case of maximal-connected subsets, an ordering of the minimum-weight simulation sequences for the maximal-connected subsets will form a minimum-weight simulation sequence.

The only change required is in the algorithm of Section 7.2. This algorithm uses a selective search technique. The reader will note that the optimal choice criteria is still applicable. The potential length test must, however, be replaced by a potential weighted sum test. The weighted algorithm characterization thus requires only a trivial modification.

8.3 Random Variables

Random variables are frequently employed in simulation problems. In the system model it was assumed that each function is deterministic. In this section the modelling of random variables is discussed.

Associated with each random variable is a function h such that successive applications of the function h will yield a set of random values in the desired distribution. Each application of the function will result in a different value.

The random variable is assumed to be distributed over simulated time. It, therefore, has only a single value associated with each time step, and all variables depending upon a random variable should use the same value for the random variable. This can be achieved by considering each random variable to be a state variable that is R-related to itself. If the random variable supplies values to variables in different processes, an output variable whose value is equal to the random variable can be added to the process. By the definition of a simulation sequence a function associated with a self-related state variable must be applied exactly once.

8.4 Time-Varying Network Structure

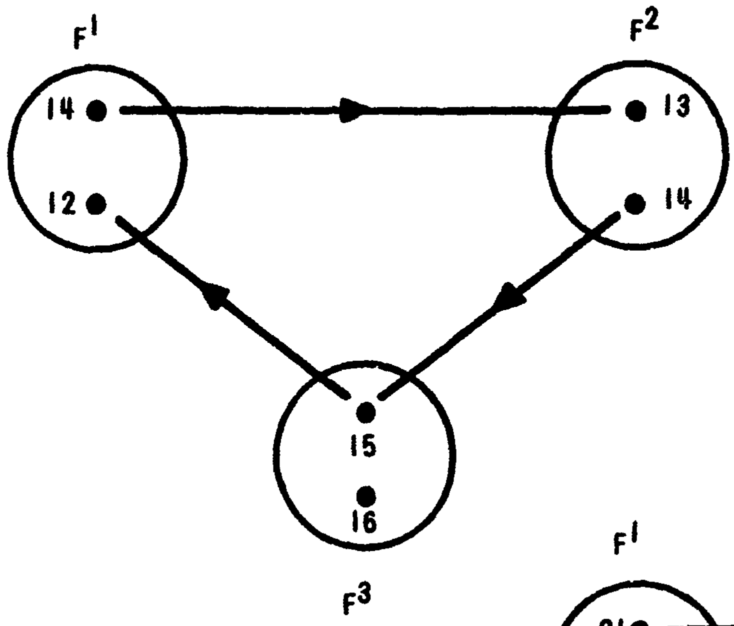
Suppose that the network structure of a simulation problem assumes two or more configurations for various time steps of the simulation and that a single simulation sequence is desired. This sequence must be a simulation sequence for each configuration of the network. In this section a technique for constructing a general simulation sequence for a multi-configuration network is presented.

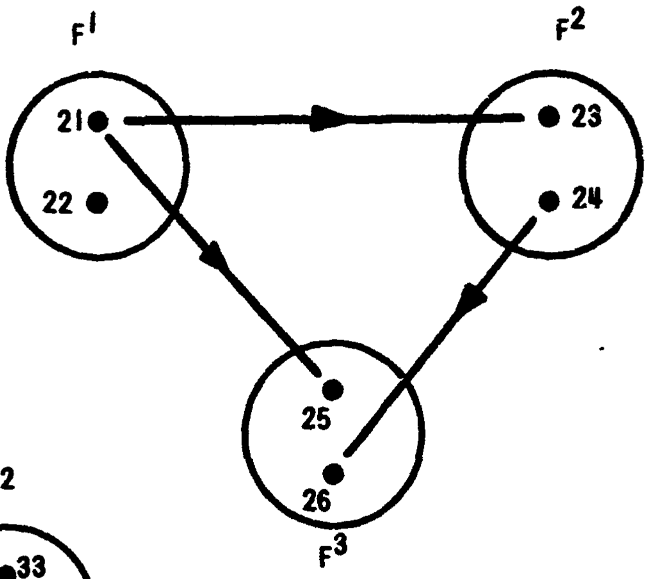
Each configuration of the network is assumed to be represented by an acyclic network graph. These graphs are assumed to be disjoint and subgraphs of a general network output graph. The general network output graph contains a set of nodes for each output. The application of an algorithm containing an output will calculate new values for each output in the set of nodes corresponding to this output. A simulation sequence for this general network output graph is a simulation sequence for each configuration of the network.

The network output graphs for three configurations of a network are shown in Figure 28. The general network output graph for this example is shown in Figure 29. The sequence $F^1F^2F^3F^1$ is a minimum-length simulation sequence for the general network output graph. This sequence is a simulation sequence for each configuration of the network.

A minimum-length simulation sequence for a general network output graph is the shortest sequence which is a simulation sequence for every configuration of the network. Some configurations may have a shorter simulation sequence. For the configuration containing outputs 21 through 26, the sequence $F^1F^2F^3$ is a simulation sequence.

The advantage of a general simulation sequence for a set of network configurations is that separate simulation sequences need not be stored for each configuration and that a selection of sequence need not be performed. The disadvantage of a general simulation sequence is that this sequence is in general not a minimum-length sequence for every configuration.





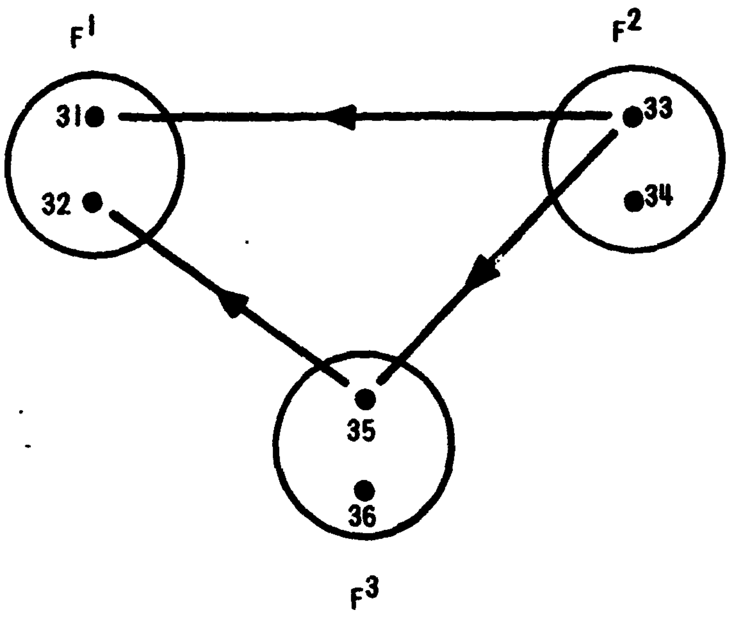


FIGURE 28

NETWORK OUTPUT GRAPHS FOR
THREE NETWORK CONFIGURATIONS

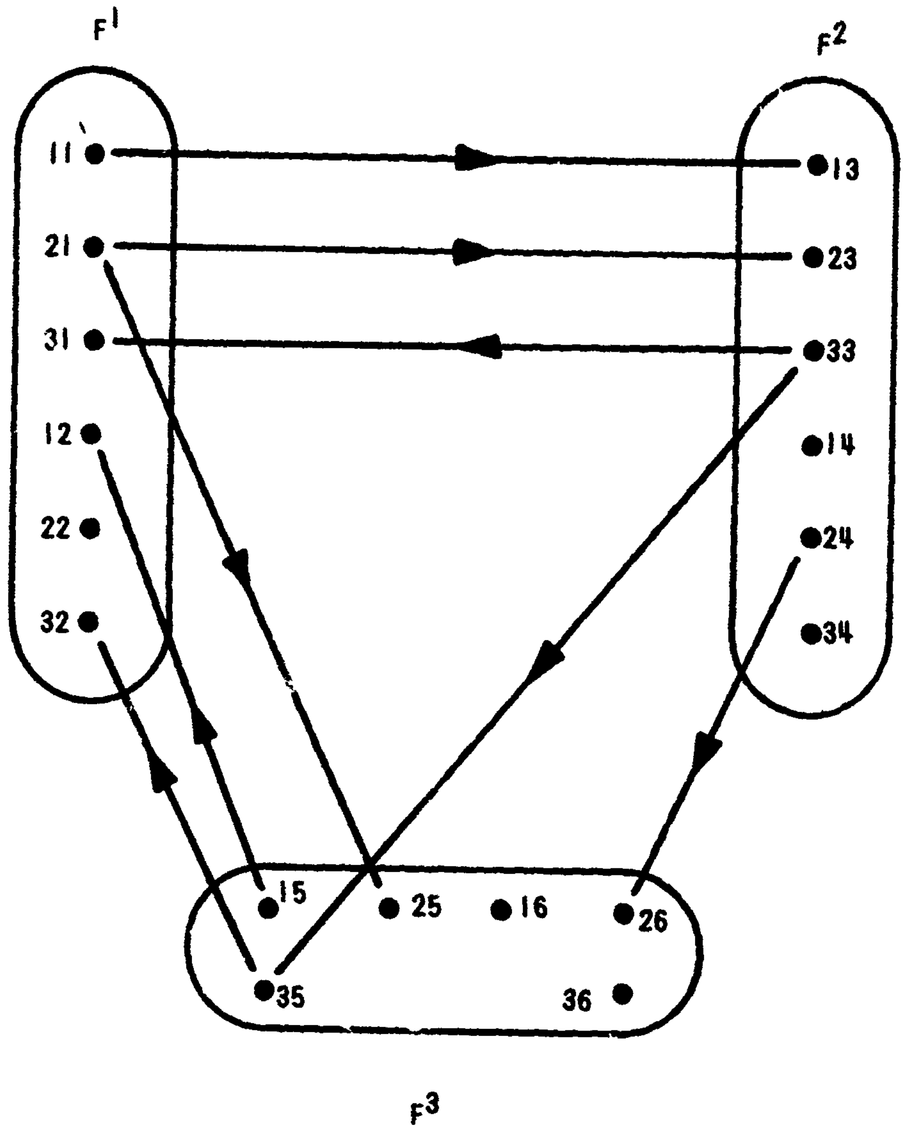


FIGURE 29

GENERAL NETWORK OUTPUT GRAPH

8.5 Time-Step Minimization

The problem of constructing a simulation sequence for a time-step has been discussed. A simulation problem generally has a large number of time-steps. In this section the reduction of the number of time-steps is discussed.

Suppose that each output has a small amount of delay associated with it and that the network is quiescent. Due to the small amount of delay associated with each output, any change of value in an input or state variable, called an event, cannot propagate instantaneously through the network but will ripple through the network. If the network is acyclic and no new event occurs, the outputs will reach a quiescent state.

In general many time steps are required before the quiescent state is reached. If the transient behavior of the network is not required, these time steps can be eliminated by calculating the quiescent state output values directly.

A quiescent value can be calculated for an output only if the outputs on which this output depends have quiescent value. This dependency relation is the same relation as the output dependency relation used in the formation of a simulation sequence. The output simulation sequence will thus calculate the quiescent state values.

The degree to which the reduction of time steps can be employed is dependent upon the output desired from the simulation and the occurrence of events within the simulation.

IX. SUMMARY AND CONCLUSIONS

9.1 General

The problem of forming a simulation sequence for networks with interactive processes has been discussed. The process interaction is specified in the system model by the process and network relations. The problem of forming a simulation sequence for the system model was reduced to the problem of forming an output simulation sequence.

An output dependency relation was defined and represented by the network output graph. A simulation sequence exists only for systems represented by an acyclic network output graph. Paths in the network output graph correspond to subsequences of simulation sequences. Partial orderings of the set of outputs were represented by the forward and backward level sets.

An algorithm-dependency relation was obtained from the output dependency relation via a condensation with respect to the algorithm definition. This relation is represented by the algorithm graph. If the relation is acyclic, it specifies a minimum-length simulation sequence. If the relation is cyclic, an ordering of the set of algorithms is not possible. An ordering of the maximal connected subsets of the algorithm graph is, however, always possible. Furthermore, a concatenation of minimum-length simulation sequences for maximal connected subsets was shown to form a minimum-length simulation sequence.

A criteria for splitting algorithms has been presented. Algorithm which are not compatible sets must be multiple included in simulation sequences and hence can be partitioned. A general level partition of an algorithm into maximal subsets and refinements of these subsets has been described.

A selective search technique for the construction of simulation sequences for problems represented by a connected algorithm graph of maximal algorithm subsets has been presented. This algorithm employs a one-step look ahead, recognizes optimal choices and includes a potential length test.

A general method for the construction of simulation sequences was described. This method uses all the techniques developed.

By means of simple modifications this method can be used to form simulation sequences for the single algorithm per process characterization of the problem. The construction of minimum-cost simulation sequences for a network with positive algorithm costs required only a change in the potential length test employed by the selective search routine.

9.2 Contributions of the Research

The contributions of this research are mainly in the area of digital computer simulations and especially relating to the simulation of systems composed of multiple-output interactive subsystems. The following list is a summary of the significant contributions:

1. The formation of simulation sequences has been shown to be independent of process state variables. Only the output dependency relation need be considered.

2. The algorithm dependency relation has been shown to partition the simulation sequence formation problem into a collection of smaller problems.

3. An algorithm splitting technique has been described. This technique greatly reduces the amount of computation in the formation of simulation sequences for many problems.

4. A general method for the formation of minimum-length simulation sequences employing (1), (2), (3) and a selective search routine has been presented.

9.3 Future Extensions

The work described in this thesis has suggested the following areas for further research.

1. Development of an analytic routine for forming simulation sequences for a connected algorithm graph composed of maximal algorithm subsets.

2. Investigation into the interaction of processes over time intervals greater than a single time-step.

3. Examination of algorithm equivalences and the examination of simulation sequence equivalences.

REFERENCES

1. Knollman, D. J. H., "Simulation Sequences For Discrete-Time Systems With Parallel Interactive Processes - An Introduction", NYU Technical Report No. 403-14, Department of Electrical Engineering, New York University, Bronx, New York, October, 1970.

2. Murphy, J. G., "A Comparison of the use of the GPSS and SIMSCRIPT Languages in Designing Communications Networks", Technical Memorandum, TN-03969, Mitre Corp., Bedford, Mass., 1964.

3. Markowitz, H., Hausner, B., and Karr, H., SIMSCRIPT: A Simulation Programming Language, Prentice-Hall Inc., 1963.

4. Gordon, G., "A General Purpose Systems Simulation Program", Proceedings of the 1961 Eastern Joint Computer Conference, pp. 87-104, 1961.

5. Knuth, D. E., and McNeley, J. L., "A Formal Definition of SOL", IEEE Trans. on EC, pp. 409-414, 1964.

   __________, "SOL - A Symbolic Language for General-Purpose System Simulation, "IEEE Trans. on EC, pp. 41-408, 1964.

6. Dahl, O. J. and Nygaard, K., "SIMULA - an ALGOL Based Simulation Language", Comm. of the ACM, Vol. 9, No. 9, pp. 671-678, 1966.

7. Gaskill, R. A., "A Versatile Problem - Oriented Language for Engineers", IEEE Trans. on EC-13, August 1964, pp. 415-421, 1964.

8. Bonine, K. C. and Hudson, R. M., "MIDAS IV A General Digital Simulation Program", GDC-ERR-AN-1043, General Dynamics, Corvair Division RD-1 No. 111-1298-911, 1967.

9. Kelly, J. L., Lockbaum, C., and Vyssotsky, V. A., "A Block Diagram Compiler," Bell System Technical Journal, pp. 669-676, May 1961.

10. Parnas, D. L., "On Simulating Networks of Parallel Processes in which Simultaneous Events may Occur", Comm. of the ACM, Vol. 12, No. 9, pp. 519-531, 1969.

11. Warshall, S., "A Theorem on Boolean Matrices", Journal of the ACM Vol. 9, pp. 11-12, 1962.

12. Harary, F., Norman, R. Z., and Cartwright, D., Structural Models: An Introduction to the Theory of Directed Graphs, John Wiley & Sons Inc., New York, 1965.

13. Marimont, R. B., "A New Method of Checking the Consistency of Precedence Matrices", Journal of the ACM, Vol. 6, pp. 164-171, 1959.

14. Ramamoorthy, C. V., "Analysis of Graphs for Connectivity Considerations," Journal of the ACM Vol. 13, No. 2, pp. 211-222, 1966.

APPENDIX

User Bulletin
Date: March 19, 1971
Prepared by: Dieter Knollman
Applicable to: Univac 1108
Type: Software

New York University
Department of Electrical Engineering
Computer Bulletin

SIMULATION SEQUENCE FORMATION PROGRAM

I. ABSTRACT

This program will form a minimum-length simulation sequence for an interactive discrete-time system. The system is represented by a block diagram. The input to the program consists of a specification of the output dependency relation and a listing of outputs included in each algorithm. The output of the program is a minimum-length string of algorithms that will accurately simulate the behavior of the system.

II. PERFORMANCE SPECIFICATION

The program can not form simulation sequences for problems with cyclic output dependencies. For these problems a listing of all cyclic output dependencies is produced.

The program requires that all outputs and algorithms are numbered in consecutive order using one-indexing. Empty algorithms are allowed. Each output must be included in a single algorithm. If this requirement is violated, the program will produce a list of outputs that are included in no algorithm and a list of outputs that are included in two or more algorithms.

The speed of the program is problem dependent. The core requirements are determined by the number of outputs. The parameter IPA determines the size of data arrays. This parameter must be greater than the number of outputs. In the program IPA is equal to 50. IPA can be altered by changing a single parameter definition statement.

III. DESCRIPTION

The program consists of a collection of external subroutines and a main program. The main program determines the storage allocation and schedules the execution of subroutines. A description of the main program and the subroutines is presented in the listing.

IV. OPERATING INSTRUCTIONS

The necessary control cards to compile and execute the program are supplied with the program. The user must supply a RUN card and data cards. The algorithm definition cards must precede the output dependency definition cards. The format of the data cards is presented in the listing of the READA and the READR subroutines.

V. EXAMPLE

A typical run will consist of the following:

RUN Card

FORTRAN V deck with control cards

Algorithm definition data cards

Output dependency data cards

EOF card

VITA

Dieter John Henry Knollman was born May 25, 1941 in Buende, West Germany. He immigrated to the United States in 1952. He is a graduate of Maury High School, Norfolk, Virginia, class of 1959. He attended Virginia Polytechnic Institute and received a Bachelor of Science Degree in Electrical Engineering. Upon graduation he became a member of the technical staff of Bell Telephone Laboratories, Holmdel, New Jersey. Under Bell Telephone Laboratories Graduate Study Program he attended the University of Illinois and received a Master of Science degree in Electrical Engineering From September 1965 to September 1969 he attended New York University on a part-time basis.

At Bell Telephone Laboratories he has been engaged in the exploratory design of Key Telephone Systems. He has been granted one patent and has four patents pending. From September 1969 to January 1971 he attended New York University on a full time basis and performed research in the formation of simulation sequences. At present he is engaged by Bell Telephone Laboratories in Denver, Colorado.